Ruby - Bug #20154

aarch64: configure overrides `-mbranch-protection` if it was set in CFLAGS via environment

01/05/2024 09:25 PM - jprokop (Jarek Prokop)

Status:	Closed		
Priority:	Normal		
Assignee:	kjtsanaktsidis (KJ Tsanaktsidis)		
Target version:			
ruby -v:	ruby 3.3.0 (2023-12-25 revision 5124f9ac75) [aarch64-linux]	**Backport:**	3.0: UNKNOWN, 3.1: UNKNOWN, 3.2: UNKNOWN, 3.3: UNKNOWN

Description

Recently a GH PR was merged https://github.com/ruby/ruby/pull/9306 For PAC/BTI support on ARM CPUs for Coroutine.S.

Without proper compilation support in configure.ac it segfaults Ruby with fibers on CPUs where PAC is supported:
https://bugs.ruby-lang.org/issues/20085

At the time of writing, configure.ac appends the first option from a list for flag -mbranch-protection that successfully compiles a program https://github.com/ruby/ruby/blob/master/configure.ac#L829,
to XCFLAGS and now also ASFLAGS to fix issue 20085 for Ruby master.

This is suboptimal for Fedora as we set -mbranch-protection=standard by default in C{,XX}FLAGS:

```
CFLAGS='-O2 -flto=auto -ffat-lto-objects -fexceptions -g -grecord-gcc-switches -pipe -Wall -Werror
=format-security -Werror=implicit-function-declaration -Werror=implicit-int -Wp,-U_FORTIFY_SOURCE,
-D_FORTIFY_SOURCE=3 -Wp,-D_GLIBCXX_ASSERTIONS -specs=/usr/lib/rpm/redhat/redhat-hardened-cc1 -fsta
ck-protector-strong -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1  -mbranch-protection=standard -f
asynchronous-unwind-tables -fstack-clash-protection -fno-omit-frame-pointer -mno-omit-leaf-frame-p
ointer '
export CFLAGS
CXXFLAGS='-O2 -flto=auto -ffat-lto-objects -fexceptions -g -grecord-gcc-switches -pipe -Wall -Werr
or=format-security -Wp,-U_FORTIFY_SOURCE,-D_FORTIFY_SOURCE=3 -Wp,-D_GLIBCXX_ASSERTIONS -specs=/usr
/lib/rpm/redhat/redhat-hardened-cc1 -fstack-protector-strong -specs=/usr/lib/rpm/redhat/redhat-ann
obin-cc1  -mbranch-protection=standard -fasynchronous-unwind-tables -fstack-clash-protection -fno-
omit-frame-pointer -mno-omit-leaf-frame-pointer'
export CXXFLAGS
```

And the appended flag overrides distribution's compilation configuration, which in this case ends up omitting BTI instructions and only using PAC.

Would it make sense to check if such flags exist and not overwrite them if they do?

Serious proposals:

1. Simplest fix that does not overwrite what is set in the distribution and results in higher security is simply prepending the list of options with -mbranch-protection=standard, it should cause no problems on ARMv8 CPUs and forward, BTI similarly to PAC instructions result into NOP, it is only extending the capability.

See attached 0001-aarch64-Check-mbranch-protection-standard-first-to-u.patch

1. Other fix that sounds more sane IMO and dodges this kind of guessing where are all the correct places for the flag is what another Fedora contributor Florian Weimer suggested:
https://lists.fedoraproject.org/archives/list/devel@lists.fedoraproject.org/message/CVTNF2OQCL3XZHUUFNYMDK6ZEF2SWUEN/

"The reliable way to do this would be to compile a C file and check
whether that enables __ARM_FEATURE_PAC_DEFAULT, and if that's the case,
define a *different* macro for use in the assembler implementation.
This way, you don't need to care about the exact name of the option."

IOW instead of using *_ARM_FEATURE** directly in that code, define a macro in the style of "USE_PAC" with value of the feature if it is defined, I think that way we shouldn't need to append ASFLAGS anymore.

However it's also important to catch the value of those macros as their values have meaning, I have an idea how to do that but I'd get on that monday earliest.

Associated revisions

Revision 0ccb80d6bf57cd6e79ad622c024d3d0940ec6f3b - 06/11/2024 10:48 AM - KJ Tsanaktsidis

Extract hardening CFLAGS to a special $hardenflags variable

This changes the automatic detection of -fstack-protector,
-D_FORTIFY_SOURCE, and -mbranch-protection to write to $hardenflags
instead of $XCFLAGS. The definition of $cflags is changed to
"$hardenflags $orig_cflags $optflags $debugflags $warnflags" to match.

Furthermore, these flags are *prepended* to $hardenflags, rather than
appended.

The implications of doing this are as follows:

- If a CRuby builder specifies cflags="-mbranch-protection=foobar" at the ./configure script, and the configure script detects that -mbranch-protection=pac-ret is accepted, then GCC will be invoked as "gcc -mbranch-protection=pac-ret -mbranch-protection=foobar". Since the last flags take precedence, that means that user-supplied values of these flags in $cflags will take priority.
- Likewise, if a CRuby builder explicitly specifies "hardenflags=-mbranch-protection=foobar", because we *prepend* to $hardenflags in our autoconf script, we will still invoke GCC as "gcc -mbranch-protection=pac-ret -mbranch-protection=foobar".
- If a CRuby builder specifies CFLAGS="..." at the configure line, automatic detection of hardening flags is ignored as before.
- C extensions will *also* be built with hardening flags now as well (this was not the case by default before because the detected flags went into $XCFLAGS).

Additionally, as part of this work, I changed how the detection of PAC/BTI in Context.S works. Rather than appending the autodetected option to ASFLAGS, we simply compile a set of test programs with the actual CFLAGS in use to determine what PAC/BTI settings were actually chosen by the builder. Context.S is made aware of these choices through some custom macros.

The result of this work is that:

- Ruby will continue to choose some sensible defaults for hardening options for the C compiler
- Distributors are able to specify CFLAGS that are consistent with their distribution and override these defaults
- Context.S will react to whatever -mbranch-protection is actually in use, not what was autodetected
- Extensions get built with hardening flags too.

[Bug #20154]
[Bug #20520]

Revision 0ccb80d6bf57cd6e79ad622c024d3d0940ec6f3b - 06/11/2024 10:48 AM - KJ Tsanaktsidis

Extract hardening CFLAGS to a special $hardenflags variable

This changes the automatic detection of -fstack-protector,
-D_FORTIFY_SOURCE, and -mbranch-protection to write to $hardenflags
instead of $XCFLAGS. The definition of $cflags is changed to
"$hardenflags $orig_cflags $optflags $debugflags $warnflags" to match.

Furthermore, these flags are *prepended* to $hardenflags, rather than
appended.

The implications of doing this are as follows:

- If a CRuby builder specifies cflags="-mbranch-protection=foobar" at the ./configure script, and the configure script detects that -mbranch-protection=pac-ret is accepted, then GCC will be invoked as "gcc -mbranch-protection=pac-ret -mbranch-protection=foobar". Since the last flags take precedence, that means that user-supplied values of these flags in $cflags will take priority.

- Likewise, if a CRuby builder explicitly specifies "hardenflags=-mbranch-protection=foobar", because we *prepend* to $hardenflags in our autoconf script, we will still invoke GCC as "gcc -mbranch-protection=pac-ret -mbranch-protection=foobar".
- If a CRuby builder specifies CFLAGS="..." at the configure line, automatic detection of hardening flags is ignored as before.
- C extensions will *also* be built with hardening flags now as well (this was not the case by default before because the detected flags went into $XCFLAGS).

Additionally, as part of this work, I changed how the detection of PAC/BTI in Context.S works. Rather than appending the autodetected option to ASFLAGS, we simply compile a set of test programs with the actual CFLAGS in use to determine what PAC/BTI settings were actually chosen by the builder. Context.S is made aware of these choices through some custom macros.

The result of this work is that:

- Ruby will continue to choose some sensible defaults for hardening options for the C compiler
- Distributors are able to specify CFLAGS that are consistent with their distribution and override these defaults
- Context.S will react to whatever -mbranch-protection is actually in use, not what was autodetected
- Extensions get built with hardening flags too.

[Bug #20154]
[Bug #20520]

Revision 0ccb80d6 - 06/11/2024 10:48 AM - KJ Tsanaktsidis

Extract hardening CFLAGS to a special $hardenflags variable

This changes the automatic detection of -fstack-protector, -D_FORTIFY_SOURCE, and -mbranch-protection to write to $hardenflags instead of $XCFLAGS. The definition of $cflags is changed to "$hardenflags $orig_cflags $optflags $debugflags $warnflags" to match.

Furthermore, these flags are *prepended* to $hardenflags, rather than appended.

The implications of doing this are as follows:

- If a CRuby builder specifies cflags="-mbranch-protection=foobar" at the ./configure script, and the configure script detects that -mbranch-protection=pac-ret is accepted, then GCC will be invoked as "gcc -mbranch-protection=pac-ret -mbranch-protection=foobar". Since the last flags take precedence, that means that user-supplied values of these flags in $cflags will take priority.
- Likewise, if a CRuby builder explicitly specifies "hardenflags=-mbranch-protection=foobar", because we *prepend* to $hardenflags in our autoconf script, we will still invoke GCC as "gcc -mbranch-protection=pac-ret -mbranch-protection=foobar".
- If a CRuby builder specifies CFLAGS="..." at the configure line, automatic detection of hardening flags is ignored as before.
- C extensions will *also* be built with hardening flags now as well (this was not the case by default before because the detected flags went into $XCFLAGS).

Additionally, as part of this work, I changed how the detection of PAC/BTI in Context.S works. Rather than appending the autodetected option to ASFLAGS, we simply compile a set of test programs with the actual CFLAGS in use to determine what PAC/BTI settings were actually chosen by the builder. Context.S is made aware of these choices through some custom macros.

The result of this work is that:

- Ruby will continue to choose some sensible defaults for hardening options for the C compiler
- Distributors are able to specify CFLAGS that are consistent with their distribution and override these defaults
- Context.S will react to whatever -mbranch-protection is actually in use, not what was autodetected

- Extensions get built with hardening flags too.

[Bug #20154]
[Bug #20520]

History

#1 - 01/05/2024 09:27 PM - jprokop (Jarek Prokop)

- ruby -v set to ruby 3.3.0 (2023-12-25 revision 5124f9ac75) [aarch64-linux]

I have looked at other aspects of the options and inspected the assembly outputted with -mbranch-protection={standard,pac-ret} in Fedora PR's workaround https://src.fedoraproject.org/rpms/ruby/pull-request/167

#2 - 01/07/2024 06:10 PM - katei (Yuta Saito)

Right, the configure.ac should respect user given CFLAGS as much as possible.
I will make follow-up fixes to avoid overriding the option, but I don't think we can get it merged in 3.3.x branch since the overriding issue is not an obvious regression. (3.2.x also has the same problem, IIUC)

Anyway, your patch in the downstream seems reasonable to me as a temporal fix.

#3 - 04/23/2024 03:27 PM - vo.x (Vit Ondruch)

Any update please? We are still carrying downstream patch in Fedora, so it would be nice to get this upstreamed

#4 - 05/18/2024 02:59 PM - kjtsanaktsidis (KJ Tsanaktsidis)

I don't think I quite understand what exactly the right course of action here is.

> Would it make sense to check if such flags exist and not overwrite them if they do?

Why is this not a serious proposal? "We should respect user given CFLAGS as much as possible" sounds like a good philosophy, so this seems sensible? (I thought about just compiling a test program and seeing if it had __ARM_FEATURE_PAC_DEFAULT/__ARM_FEATURE_BTI_DEFAULT set, but that would not be able to tell if the user explicitly provided -mbranch-protection=none, and *that* also should be respected).

> Other fix that sounds more sane IMO and dodges this kind of guessing where are all the correct places for the flag is what another Fedora contributor Florian Weimer suggested

I'm not 100% sure, but is Florian suggesting here that:

- passing -mbranch-protector= to the assembler doesn't actually *do* anything other than defining the macros,
- So our build system should just define its own macros by doing something like AC_CHECK_DECLS(__ARM_FEATURE_PAC_DEFAULT) etc and defining RUBY_AARCH64_PAC etc as a result
- And the assembly files should check RUBY_AARCH64_PAC instead of __ARM_FEATURE_PAC_DEFAULT
- And that this is better because distributions etc will put -mbranch-protection=pac-ret+bti in their CFLAGS, but not normally in their ASFLAGS (because this does nothing anyway)

However doing some research it seems that other projects also apply their branch protection options to ASFLAGS e.g. https://source.chromium.org/chromium/chromium/src/+/main:build/config/linux/BUILD.gn;l=23. So i'm not super sure about this. Should distros not also put -mbranch-protection in their ASFLAGS if they want them system-wide? (And also for e.g. rust https://doc.rust-lang.org/beta/unstable-book/compiler-flags/branch-protection.html)

> However it's also important to catch the value of those macros as their values have meaning

It seems that based on https://developer.arm.com/documentation/101028/0012/5--Feature-test-macros?lang=en that

- __ARM_FEATURE_BTI_DEFAULT is either undefined or 1 if BTI is enabled, so we don't need its meaning
- __ARM_FEATURE_PAC_DEFAULT bits give a meaning though, and our current assembly in Coroutine.S totally ignores these meanings
  - Bits 0 and 1 determine which key to use. At the moment we sign with the A-key only. I don't know why PAC has two keys, but if a distro sets -mbranch-protection=pac-ret+b-key, we're going to be ignoring that.
  - Bit 2 determines if leaf functions that don't set up a stack frame should be protected. coroutine_transfer should always be protected so i don't think we need any information from this bit
  - Bit 3 determines if we should also add PC to the signature (I think? it means "Protection using PC as a diversifier"). This is for the FEAT_PAuth_LR extension which is new (https://github.com/ARM-software/acle/pull/292) and I don't know if any hardware actually has it yet? I don't know if this is something we would need to implement for coroutine_transfer or not. Maybe it wouldn't even make sense because the whole point of coroutine_transfer is to return to a different place than it was called from.

Maybe we should just #error if the B-key is selected and leave it at that; if someone has a use-case then it can be implemented?

I've written a lot of text here but it boils down to "I'm not sure what we should do". Perhaps we should rip out *all* branch-protection flags out of

configure.ac and leave it to distributors to specify the CFLAGS/ASFLAGS that they want?

#5 - 06/06/2024 07:32 AM - kjtsanaktsidis (KJ Tsanaktsidis)

- Assignee set to kjtsanaktsidis (KJ Tsanaktsidis)

#6 - 06/06/2024 02:37 PM - vo.x (Vit Ondruch)

BTW since this is about overriding configuration options, I think that #20520 falls into the same bucket

#7 - 06/07/2024 03:29 AM - kjtsanaktsidis (KJ Tsanaktsidis)

Ah yeah, I did see that - I'll try and tackle these two together.

In general it seems that the CRuby build system is perhaps quite inflexible for distributors like yourself who want precise control over compilation flags and what not.

While I was trying to figure out the guts of Ruby's autoconf script yesterday, I had a couple of questions which came to mind, that you might be able to help answer...

- From a distro-packager perspective, how do you expect CFLAGS vs XCFLAGS to behave? When you set CFLAGS=..., do you expect those to carry over to native gems installed with gem install? Are there ever flags you want to apply to Ruby that you don't want to apply to gems? Or do you just not care at all because you package gems in separate RPM's with your own CFLAGS there too, and nobody should ever gem install with a distro-provided Ruby?
- Do you know how other autoconf-based programs handle optimistically turning on things like -DFORTIFY_SOURCE or -mbranch-protection by default? It seems to me that a lot of problems would go away if we prepended stuff like this to the user-provided flags, rather than appended it (since gcc will just pick the last one). Is that normal, do you think?

I'll go assign the fortify_source ticket to myself too, I'm hoping to get some more time this weekend to poke at it.

#8 - 06/07/2024 03:34 AM - kjtsanaktsidis (KJ Tsanaktsidis)

And two more questions:

- It seems that if you build with CFLAGS=, you totally overwrite Ruby's default optimisation flags & warning flags, whereas if you use cflags=, you prepend to them (or you can specifically overwrite optflags= etc). I guess you build with CFLAGS= because you really do want to decide for yourself what the compilation flags should be including all the warnings etc - is that right?
- Part of the problem seems to be that Ruby's configure script puts stuff like -mbranch-protection in XCFLAGS and then builds with gcc $CFLAGS $XCFLAGS. I guess it would be more normal if we built with gcc $XCFLAGS $CFLAGS instead, wouldn't it?

#9 - 06/07/2024 11:07 AM - vo.x (Vit Ondruch)

kjtsanaktsidis (KJ Tsanaktsidis) wrote in #note-7:

> Ah yeah, I did see that - I'll try and tackle these two together.

> - From a distro-packager perspective, how do you expect CFLAGS vs XCFLAGS to behave? When you set CFLAGS=..., do you expect those to carry over to native gems installed with gem install? Are there ever flags you want to apply to Ruby that you don't want to apply to gems? Or do you just not care at all because you package gems in separate RPM's with your own CFLAGS there too, and nobody should ever gem install with a distro-provided Ruby?

I wish we could ignore gem install, but there is too many gems around (and Bundler). That leaves us in unfortunate place, because we struggle with issues such as this.

IOW for RPM, we are setting the flags and configure options like this:

https://github.com/rpm-software-management/rpm/blob/8ab304f173379e539329aaf528920f50cee68638/macros.in#L1017-L1051

What e.g. configure detects on build system does not match with user installation. It might be difference in tools / libraries installed but also different HW which would benefit from different optimizations.

> - Do you know how other autoconf-based programs handle optimistically turning on things like -DFORTIFY_SOURCE or -mbranch-protection by default?

Unfortunately, I am hardly expert on autoconf. In general, I am fan of optimistically enabling features such as -DFORTIFY_SOURCE. After all, Fedora might just benefit if this kind of features are already tested / supported upstream. OTOH, Fedora sometimes leads in those effort as in this case.

> It seems to me that a lot of problems would go away if we prepended stuff like this to the user-provided flags, rather than appended it (since gcc will just pick the last one). Is that normal, do you think?

I am hardly expert on compiler command line. For me it would be certainly easier, if I saw each flag just one time. But prepending might generally provide results close to expected state. I think this is also my answer to your gcc $XCFLAGS $CFLAGS question.

> It seems that if you build with CFLAGS=, you totally overwrite Ruby's default optimisation flags & warning flags, whereas if you use cflags=, you prepend to them (or you can specifically overwrite optflags= etc). I guess you build with CFLAGS= because you really do want to decide for yourself what the compilation flags should be including all the warnings etc - is that right?

The consistency of compilation options is one of the advantages of distributions such as Fedora. However there is more, e.g. supporting multiple architectures. I just trust the toolchain folks that the default set is the best for Fedora.

But we might be missing some flags which would be beneficial for Ruby in Fedora. I am happy to learn about those.

#10 - 06/09/2024 11:50 PM - kjtsanaktsidis (KJ Tsanaktsidis)

https://github.com/ruby/ruby/pull/10944 - would this solve the problem for you?

#11 - 06/11/2024 11:05 AM - Anonymous

- Status changed from Open to Closed

Applied in changeset git|0ccb80d6bf57cd6e79ad622c024d3d0940ec6f3b.

Extract hardening CFLAGS to a special $hardenflags variable

This changes the automatic detection of -fstack-protector,
-D_FORTIFY_SOURCE, and -mbranch-protection to write to $hardenflags
instead of $XCFLAGS. The definition of $cflags is changed to
"$hardenflags $orig_cflags $optflags $debugflags $warnflags" to match.

Furthermore, these flags are *prepended* to $hardenflags, rather than
appended.

The implications of doing this are as follows:

- If a CRuby builder specifies cflags="-mbranch-protection=foobar" at the ./configure script, and the configure script detects that -mbranch-protection=pac-ret is accepted, then GCC will be invoked as "gcc -mbranch-protection=pac-ret -mbranch-protection=foobar". Since the last flags take precedence, that means that user-supplied values of these flags in $cflags will take priority.
- Likewise, if a CRuby builder explicitly specifies "hardenflags=-mbranch-protection=foobar", because we *prepend* to $hardenflags in our autoconf script, we will still invoke GCC as "gcc -mbranch-protection=pac-ret -mbranch-protection=foobar".
- If a CRuby builder specifies CFLAGS="..." at the configure line, automatic detection of hardening flags is ignored as before.
- C extensions will *also* be built with hardening flags now as well (this was not the case by default before because the detected flags went into $XCFLAGS).

Additionally, as part of this work, I changed how the detection of
PAC/BTI in Context.S works. Rather than appending the autodetected
option to ASFLAGS, we simply compile a set of test programs with the
actual CFLAGS in use to determine what PAC/BTI settings were actually
chosen by the builder. Context.S is made aware of these choices through
some custom macros.

The result of this work is that:

- Ruby will continue to choose some sensible defaults for hardening options for the C compiler
- Distributors are able to specify CFLAGS that are consistent with their distribution and override these defaults
- Context.S will react to whatever -mbranch-protection is actually in use, not what was autodetected
- Extensions get built with hardening flags too.

[Bug #20154]
[Bug #20520]

Files

0001-aarch64-Check-mbranch-protection-standard-first-to-u.patch 1004 Bytes 01/05/2024 jprokop (Jarek Prokop)