Ruby master - Feature #10351

[PATCH] prevent CVE-2014-6277

10/09/2014 03:44 PM - shyouhei (Shyouhei Urabe)

Status:	Feedback
Priority:	Normal
Assignee:	
Target version:	

Description

```
From 4636ca0308f1933c9b191f36e808a8d3bcf5e88e Mon Sep 17 00:00:00 2001
From: "Urabe, Shyouhei" <shyouhei@ruby-lang.org>
Date: Wed, 8 Oct 2014 15:44:27 +0900
Subject: [PATCH] prevent CVE-2014-6277

ShellShock was about bash.  I think ruby is torellant for that kind of
attack.  However the concept was that environment variables can
potentially contain malicious data.  To pass them to victim
subprocecsses can sometimes cause catastrophic situation, like
arbitrary code execution.  Even though ruby itself is not affected, it
can be an accomplice in attacks by blindly passing through any
uncontrolled info through.  Let's just change this, more secure by
default.

This patch does not add a new feature, nor delete anything.  It just
changes the default behaviour when ruby spawns subprocesses.

    Process.spawn('/usr/bin/printenv') # -> prints nothing

You can explicitly pass what you need:

    Process.spawn({'FOO'=>'BAR'}, '/usr/bin/printenv')

Or if you are 128% sure what you are doing, can pass everything.

    ENV['FOO'] = 'BAR'
    Process.spawn('/usr/bin/printenv', unsetenv_others: false)

I don't intend to make things impossible; just give it a better
default.  It doesn't ultimately solve everything but it should prevent
casual faults.

---
 process.c | 8 ++++----
 1 file changed, 4 insertions(+), 4 deletions(-)

diff --git a/process.c b/process.c
index f9bc01c..395333d 100644
--- a/process.c
+++ b/process.c
@@ -2295,7 +2295,7 @@ rb_execarg_fixup(VALUE execarg_obj)
         eargp->dup2_tmpbuf = tmpbuf;
     }

-    unsetenv_others = eargp->unsetenv_others_given && eargp->unsetenv_others_do;
+    unsetenv_others = !eargp->unsetenv_others_given || eargp->unsetenv_others_do;
     envopts = eargp->env_modification;
     if (unsetenv_others || envopts != Qfalse) {
         VALUE envtbl, envp_str, envp_buf;
@@ -2936,7 +2936,7 @@ rb_execarg_run_options(const struct rb_execarg *eargp, struct rb_execarg *sa
rgp,
 #endif

 #if !defined(HAVE_WORKING_FORK)
```

```
-    if (eargp->unsetenv_others_given && eargp->unsetenv_others_do) {
+    if (!eargp->unsetenv_others_given || eargp->unsetenv_others_do) {
         save_env(sargp);
         rb_env_clear();
     }
@@ -3998,8 +3998,8 @@ rb_f_system(int argc, VALUE *argv)
  *      [cmdname, argv0], arg1, ... : command name, argv[0] and zero or more arguments (no shell)
  *    options: hash
  *      clearing environment variables:
- *        :unsetenv_others => true   : clear environment variables except specified by env
- *        :unsetenv_others => false  : don't clear (default)
+ *        :unsetenv_others => true   : clear environment variables except specified by env (defau
lt)
+ *        :unsetenv_others => false  : don't clear
  *      process group:
  *        :pgroup => true or 0 : make a new process group
  *        :pgroup => pgid      : join to specified process group
--
1.9.1
```

History

#1 - 10/09/2014 05:02 PM - akr (Akira Tanaka)

It is very aggressive.

At least, you should try "make check".

```
% make check
...
./miniruby -I./lib -I. -I.ext/common  ./tool/runruby.rb --extout=.ext  -- --disable-gems "./test/runner.rb" --
ruby="./miniruby -I./lib -I. -I.ext/common  ./tool/runruby.rb --extout=.ext  -- --disable-gems"
mkmf.rb can't find header files for ruby at /home/ruby/tst1/lib/ruby/include/ruby.h
uncommon.mk:549: recipe for target 'yes-test-all' failed
make: *** [yes-test-all] Error 1
```

#2 - 10/09/2014 05:22 PM - normalperson (Eric Wong)

shyouhei@ruby-lang.org wrote:

> This patch does not add a new feature, nor delete anything. It just
> changes the default behaviour when ruby spawns subprocesses.
>
> ```
> Process.spawn('/usr/bin/printenv') # -> prints nothing
> ```

The potential for breakage is way too high. Losing some envs (e.g.
PATH, TMPDIR, SHELL or HOME) can be disastrous and introduce new
security problems.

Right now, everybody knows about shellshock and patching bash.
This is an over-reaction which causes needless breakage.

(Especially since your example never even spawns a shell)

#3 - 10/09/2014 10:38 PM - akr (Akira Tanaka)

- Status changed from Open to Feedback

#4 - 10/10/2014 03:11 AM - shyouhei (Shyouhei Urabe)

Akira, isn't it obvious this patch introduce a backward incompatibility?

I'd be surprised if it could passed the tests, because that means our tests are insufficient. Is it that important to pass make check at this point?

I don't think it's an "at least" thing for this rather conceptual patch. Is it your only concern?

Of course tests shall be modified to fit this when it is going to land.

On 10/10/2014 02:02 AM, akr@fsij.org wrote:

> Issue #10351 has been updated by Akira Tanaka.

It is very aggressive.

At least, you should try "make check".

```
% make check
...
./miniruby -I./lib -I. -I.ext/common  ./tool/runruby.rb --extout=.ext  -- --disable-gems "./test/runner.rb
" --ruby="./miniruby -I./lib -I. -I.ext/common  ./tool/runruby.rb --extout=.ext  -- --disable-gems"
mkmf.rb can't find header files for ruby at /home/ruby/tst1/lib/ruby/include/ruby.h
uncommon.mk:549: recipe for target 'yes-test-all' failed
make: *** [yes-test-all] Error 1
```

Feature #10351: [PATCH] prevent CVE-2014-6277
https://bugs.ruby-lang.org/issues/10351#change-49322

- Author: Shyouhei Urabe
- Status: Open
- Priority: Normal
- Assignee:
- Category: core

* Target version: current: 2.2.0

```
>From 4636ca0308f1933c9b191f36e808a8d3bcf5e88e Mon Sep 17 00:00:00 2001
From: "Urabe, Shyouhei" <shyouhei@ruby-lang.org>
Date: Wed, 8 Oct 2014 15:44:27 +0900
Subject: [PATCH] prevent CVE-2014-6277

ShellShock was about bash.  I think ruby is torellant for that kind of
attack.  However the concept was that environment variables can
potentially contain malicious data.  To pass them to victim
subprocecsses can sometimes cause catastrophic situation, like
arbitrary code execution.  Even though ruby itself is not affected, it
can be an accomplice in attacks by blindly passing through any
uncontrolled info through.  Let's just change this, more secure by
default.

This patch does not add a new feature, nor delete anything.  It just
changes the default behaviour when ruby spawns subprocesses.

    Process.spawn('/usr/bin/printenv') # -> prints nothing

You can explicitly pass what you need:

    Process.spawn({'FOO'=>'BAR'}, '/usr/bin/printenv')

Or if you are 128% sure what you are doing, can pass everything.

    ENV['FOO'] = 'BAR'
    Process.spawn('/usr/bin/printenv', unsetenv_others: false)

I don't intend to make things impossible; just give it a better
default.  It doesn't ultimately solve everything but it should prevent
casual faults.

---
 process.c | 8 ++++----
 1 file changed, 4 insertions(+), 4 deletions(-)

diff --git a/process.c b/process.c
index f9bc01c..395333d 100644
--- a/process.c
+++ b/process.c
@@ -2295,7 +2295,7 @@ rb_execarg_fixup(VALUE execarg_obj)
         eargp->dup2_tmpbuf = tmpbuf;
     }

-    unsetenv_others = eargp->unsetenv_others_given && eargp->unsetenv_others_do;
+    unsetenv_others = !eargp->unsetenv_others_given || eargp->unsetenv_others_do;
     envopts = eargp->env_modification;
     if (unsetenv_others || envopts != Qfalse) {
         VALUE envtbl, envp_str, envp_buf;
@@ -2936,7 +2936,7 @@ rb_execarg_run_options(const struct rb_execarg *eargp, struct rb_execarg *sargp,
 #endif
```

```
 #if !defined(HAVE_WORKING_FORK)
-    if (eargp->unsetenv_others_given && eargp->unsetenv_others_do) {
+    if (!eargp->unsetenv_others_given || eargp->unsetenv_others_do) {
         save_env(sargp);
         rb_env_clear();
     }
@@ -3998,8 +3998,8 @@ rb_f_system(int argc, VALUE *argv)
  *      [cmdname, argv0], arg1, ... : command name, argv[0] and zero or more arguments (no shell)
  *    options: hash
  *      clearing environment variables:
- *        :unsetenv_others => true   : clear environment variables except specified by env
- *        :unsetenv_others => false  : don't clear (default)
+ *        :unsetenv_others => true   : clear environment variables except specified by env (default)
+ *        :unsetenv_others => false  : don't clear
  *      process group:
  *        :pgroup => true or 0 : make a new process group
  *        :pgroup => pgid      : join to specified process group
```

#5 - 10/10/2014 03:31 AM - shyouhei (Shyouhei Urabe)

On 10/10/2014 02:16 AM, Eric Wong wrote:

> shyouhei@ruby-lang.org wrote:
>
> > This patch does not add a new feature, nor delete anything. It just
> > changes the default behaviour when ruby spawns subprocesses.
> >
> > ```
> > Process.spawn('/usr/bin/printenv') # -> prints nothing
> > ```
>
> The potential for breakage is way too high.

I understand this.

> Losing some envs (e.g.
> PATH, TMPDIR, SHELL or HOME) can be disastrous and introduce new
> security problems.

After shellshock I started thinking that every environment variables shall be inspected before passing to another process. There can be various ways, like introducing "the value is sane" flag to each env vars (default false) and let programms check them explicitly, for instance. The approach in this patch is to force programmers write what to pass. For instance if you think PATH shall not be clobbered you should add {'PATH'=ENV['PATH']}.

> Right now, everybody knows about shellshock and patching bash.

(I know at least one example who doesn't... the problem is that machine will not update ruby either)

> This is an over-reaction which causes needless breakage.
>
> (Especially since your example never even spawns a shell)

What if the spawned subprocess then spawns its own shell? Like I said ruby itself is immune to shellshock. That doesn't mean all the subprocess that we spawn are. Same discussion goes to our subprocesses as well. When they are not shells, that doesn't always mean they don't spawn a shell.

#6 - 10/13/2014 03:03 PM - akr (Akira Tanaka)

Shyouhei Urabe wrote:

> Akira, isn't it obvious this patch introduce a backward incompatibility?
>
> I'd be surprised if it could passed the tests, because that means our tests are insufficient. Is it that important to pass make check at this point?
>
> I don't think it's an "at least" thing for this rather conceptual patch. Is it your only concern?
>
> Of course tests shall be modified to fit this when it is going to land.

The problem is how easy/difficult to modify software to follow this change.

Trying to modify test-all is a way to estimate the difficulty.
I recommend it because it should be done anyway if this issue is accepted.

#7 - 10/18/2014 07:54 PM - shyouhei (Shyouhei Urabe)

Akira Tanaka wrote:

> Shyouhei Urabe wrote:
>
> > Akira, isn't it obvious this patch introduce a backward incompatibility?
> >
> > I'd be surprised if it could passed the tests, because that means our tests are insufficient. Is it that important to pass make check at this point?
> >
> > I don't think it's an "at least" thing for this rather conceptual patch. Is it your only concern?
> >
> > Of course tests shall be modified to fit this when it is going to land.
>
> The problem is how easy/difficult to modify software to follow this change.
>
> Trying to modify test-all is a way to estimate the difficulty.
> I recommend it because it should be done anyway if this issue is accepted.

I don't understand why you think it is a problem. Difficult? is it?

```
---
 lib/mkmf.rb                            |  7 +++++--
 lib/rake/file_utils.rb                 |  5 +++--
 lib/webrick/httpservlet/cgi_runner.rb | 10 ++++------
 test/lib/test/unit.rb                  |  5 ++++-
 test/rake/support/ruby_runner.rb       |  4 +++-
 test/rake/test_rake_file_utils.rb      | 13 +++++---------
 test/ruby/envutil.rb                   |  5 ++++-
 test/ruby/test_process.rb              | 11 ++++++-----
 test/ruby/test_require.rb              | 32 ++++++++++++++++++++++-----------
 test/ruby/test_rubyoptions.rb          | 28 ++++++++++++++++++----------
 test/rubygems/test_gem.rb              | 10 ++++++++--
 tool/runruby.rb                        |  5 ++++-
 12 files changed, 72 insertions(+), 63 deletions(-)

diff --git a/lib/mkmf.rb b/lib/mkmf.rb
index cb66d88..d5b5fe7 100644
--- a/lib/mkmf.rb
+++ b/lib/mkmf.rb
@@ -384,15 +384,18 @@ module MakeMakefile
     end
     Logging::open do
       puts command.quote
+      envp = {
+        "PATH"=>ENV["PATH"],
+      }
       if opts and opts[:werror]
         result = nil
         Logging.postpone do |log|
-          result = (system(libpath_env, command) and File.zero?(log.path))
+          result = (system(envp.merge(libpath_env), command) and File.zero?(log.path))
           ""
         end
         result
       else
-        system(libpath_env, command)
+        system(envp.merge(libpath_env), command)
       end
     end
   end
diff --git a/lib/rake/file_utils.rb b/lib/rake/file_utils.rb
index 27f4e2e..14e8686 100644
--- a/lib/rake/file_utils.rb
+++ b/lib/rake/file_utils.rb
@@ -90,10 +90,11 @@ module FileUtils
   #
   def ruby(*args, &block)
     options = (Hash === args.last) ? args.pop : {}
```

```
+    envp = (Hash === args.first) ? args.shift : {}
     if args.length > 1
-      sh(*([RUBY] + args + [options]), &block)
+      sh(envp, *([RUBY] + args + [options]), &block)
     else
-      sh("#{RUBY} #{args.first}", options, &block)
+      sh(envp, "#{RUBY} #{args.first}", options, &block)
     end
   end

diff --git a/lib/webrick/httpservlet/cgi_runner.rb b/lib/webrick/httpservlet/cgi_runner.rb
index 32ecb6f..85722d5 100644
--- a/lib/webrick/httpservlet/cgi_runner.rb
+++ b/lib/webrick/httpservlet/cgi_runner.rb
@@ -31,16 +31,14 @@ STDERR.reopen(open(err, "w"))
 len  = sysread(STDIN, 8).to_i
 dump = sysread(STDIN, len)
 hash = Marshal.restore(dump)
-ENV.keys.each{|name| ENV.delete(name) }
-hash.each{|k, v| ENV[k] = v if v }

-dir = File::dirname(ENV["SCRIPT_FILENAME"])
+dir = File::dirname(hash["SCRIPT_FILENAME"])
 Dir::chdir dir

 if ARGV[0]
   argv = ARGV.dup
-  argv << ENV["SCRIPT_FILENAME"]
-  exec(*argv)
+  argv << hash["SCRIPT_FILENAME"]
+  exec(hash, *argv)
   # NOTREACHED
 end
-exec ENV["SCRIPT_FILENAME"]
+exec hash, ENV["SCRIPT_FILENAME"]
diff --git a/test/lib/test/unit.rb b/test/lib/test/unit.rb
index c4fdf67..e527a1b 100644
--- a/test/lib/test/unit.rb
+++ b/test/lib/test/unit.rb
@@ -275,7 +275,10 @@ module Test

       class Worker
         def self.launch(ruby,args=[])
-          io = IO.popen([*ruby,
+          envp = {
+            "PATH" => ENV["PATH"],
+          }
+          io = IO.popen(envp, [*ruby,
                          "#{File.dirname(__FILE__)}/unit/parallel.rb",
                          *args], "rb+")
           new(io, io.pid, :waiting)
diff --git a/test/rake/support/ruby_runner.rb b/test/rake/support/ruby_runner.rb
index d51dd24..d35f9bd 100644
--- a/test/rake/support/ruby_runner.rb
+++ b/test/rake/support/ruby_runner.rb
@@ -18,7 +18,9 @@ module RubyRunner
   def run_ruby(option_list)
     puts "COMMAND: [#{RUBY} #{option_list.join ' '}]" if @verbose

-    Open3.popen3(RUBY, *option_list) {|inn, out, err, wait|
+    # It seems vital for some tests to pass RAKE_* envitonment variables
+    env = ENV.keys.grep(/^RAKE/).each_with_object({}){|k, r| r[k] = ENV[k] }
+    Open3.popen3(env, RUBY, *option_list) {|inn, out, err, wait|
       inn.close

       @exit = wait ? wait.value : $?
diff --git a/test/rake/test_rake_file_utils.rb b/test/rake/test_rake_file_utils.rb
index 37d33dc..80971d4 100644
--- a/test/rake/test_rake_file_utils.rb
+++ b/test/rake/test_rake_file_utils.rb
@@ -130,18 +130,16 @@ class TestRakeFileUtils < Rake::TestCase
   def test_sh_with_a_single_string_argument
     check_expansion

-    ENV['RAKE_TEST_SH'] = 'someval'
```

```
     verbose(false) {
-      sh %{#{RUBY} check_expansion.rb #{env_var} someval}
+      sh({'RAKE_TEST_SH'=>'someval'}, %{#{RUBY} check_expansion.rb #{env_var} someval})
     }
   end

   def test_sh_with_multiple_arguments
     check_no_expansion
-    ENV['RAKE_TEST_SH'] = 'someval'

     verbose(false) {
-      sh RUBY, 'check_no_expansion.rb', env_var, 'someval'
+      sh({'RAKE_TEST_SH'=>'someval'}, RUBY, 'check_no_expansion.rb', env_var, 'someval')
     }
   end

@@ -225,11 +223,9 @@ class TestRakeFileUtils < Rake::TestCase
   def test_ruby_with_a_single_string_argument
     check_expansion

-    ENV['RAKE_TEST_SH'] = 'someval'
-
     verbose(false) {
       replace_ruby {
-        ruby %{check_expansion.rb #{env_var} someval}
+        ruby({'RAKE_TEST_SH'=>'someval'}, %{check_expansion.rb #{env_var} someval})
       }
     }
   end
@@ -237,10 +233,9 @@ class TestRakeFileUtils < Rake::TestCase
   def test_ruby_with_multiple_arguments
     check_no_expansion

-    ENV['RAKE_TEST_SH'] = 'someval'
     verbose(false) {
       replace_ruby {
-        ruby 'check_no_expansion.rb', env_var, 'someval'
+        ruby({'RAKE_TEST_SH'=>'someval'}, 'check_no_expansion.rb', env_var, 'someval')
       }
     }
   end
diff --git a/test/ruby/envutil.rb b/test/ruby/envutil.rb
index f5fbb7c..b02bed3 100644
--- a/test/ruby/envutil.rb
+++ b/test/ruby/envutil.rb
@@ -47,7 +47,10 @@ module EnvUtil
       err_p.set_encoding(encoding) if err_p
     end
     c = "C"
-    child_env = {}
+    # some defaults
+    child_env = {
+      'RUBYLIB' => ENV['RUBYLIB'],
+    }
     LANG_ENVS.each {|lc| child_env[lc] = c}
     if Array === args and Hash === args.first
       child_env.update(args.shift)
diff --git a/test/ruby/test_process.rb b/test/ruby/test_process.rb
index b314b7f..56700d1 100644
--- a/test/ruby/test_process.rb
+++ b/test/ruby/test_process.rb
@@ -282,7 +282,8 @@ class TestProcess < Test::Unit::TestCase
       end
     end
     cmd << '-e' << 'puts ENV.keys.map{|e|e.upcase}'
-    IO.popen(cmd) {|io|
+    # @shyouhei's note: it seems this test intended to pass all envs.
+    IO.popen(cmd, unsetenv_others: false) {|io|
       assert_equal("PATH\n", io.read)
     }

@@ -298,11 +299,11 @@ class TestProcess < Test::Unit::TestCase
     old = ENV["hmm"]
     begin
       ENV["hmm"] = "fufu"
```

```
-       IO.popen(ENVCOMMAND) {|io| assert_match(/^hmm=fufu$/, io.read) }
+       IO.popen(ENVCOMMAND) {|io| assert_match(/^$/, io.read) }
        IO.popen([{"hmm"=>""}, *ENVCOMMAND]) {|io| assert_match(/^hmm=$/, io.read) }
        IO.popen([{"hmm"=>nil}, *ENVCOMMAND]) {|io| assert_not_match(/^hmm=/, io.read) }
        ENV["hmm"] = ""
-       IO.popen(ENVCOMMAND) {|io| assert_match(/^hmm=$/, io.read) }
+       IO.popen(ENVCOMMAND) {|io| assert_match(/^$/, io.read) }
        IO.popen([{"hmm"=>""}, *ENVCOMMAND]) {|io| assert_match(/^hmm=$/, io.read) }
        IO.popen([{"hmm"=>nil}, *ENVCOMMAND]) {|io| assert_not_match(/^hmm=/, io.read) }
        ENV["hmm"] = nil
@@ -323,11 +324,11 @@ class TestProcess < Test::Unit::TestCase
      old = ENV["hmm"]
      begin
        ENV["hmm"] = "fufu"
-       IO.popen(cmd) {|io| assert_match(/^hmm=fufu$/, io.read, message)}
+       IO.popen(cmd) {|io| assert_match(/^$/, io.read, message)}
        IO.popen({"hmm"=>""}, cmd) {|io| assert_match(/^hmm=$/, io.read, message)}
        IO.popen({"hmm"=>nil}, cmd) {|io| assert_not_match(/^hmm=/, io.read, message)}
        ENV["hmm"] = ""
-       IO.popen(cmd) {|io| assert_match(/^hmm=$/, io.read, message)}
+       IO.popen(cmd) {|io| assert_match(/^$/, io.read, message)}
        IO.popen({"hmm"=>""}, cmd) {|io| assert_match(/^hmm=$/, io.read, message)}
        IO.popen({"hmm"=>nil}, cmd) {|io| assert_not_match(/^hmm=/, io.read, message)}
        ENV["hmm"] = nil
diff --git a/test/ruby/test_require.rb b/test/ruby/test_require.rb
index 3bd327c..9e76bbf 100644
--- a/test/ruby/test_require.rb
+++ b/test/ruby/test_require.rb
@@ -118,9 +118,11 @@ class TestRequire < Test::Unit::TestCase
     env_rubypath, env_home = ENV["RUBYPATH"], ENV["HOME"]
     pathname_too_long = /pathname too long \(ignored\).*\(LoadError\)/m

-    ENV["RUBYPATH"] = "~"
-    ENV["HOME"] = "/foo" * 1024
-    assert_in_out_err(%w(-S -w test_ruby_test_require), "", [], pathname_too_long)
+    env = {
+      "RUBYPATH" => "~",
+      "HOME" => "/foo" * 1024,
+    }
+    assert_in_out_err([env] + %w(-S -w test_ruby_test_require), "", [], pathname_too_long)

   ensure
     env_rubypath ? ENV["RUBYPATH"] = env_rubypath : ENV.delete("RUBYPATH")
@@ -131,9 +133,11 @@ class TestRequire < Test::Unit::TestCase
     env_rubypath, env_home = ENV["RUBYPATH"], ENV["HOME"]
     pathname_too_long = /pathname too long \(ignored\).*\(LoadError\)/m

-    ENV["RUBYPATH"] = "~" + "/foo" * 1024
-    ENV["HOME"] = "/foo"
-    assert_in_out_err(%w(-S -w test_ruby_test_require), "", [], pathname_too_long)
+    env = {
+      "RUBYPATH" => "~" + "/foo" * 1024,
+      "HOME" => "/foo",
+    }
+    assert_in_out_err([env] + %w(-S -w test_ruby_test_require), "", [], pathname_too_long)

   ensure
     env_rubypath ? ENV["RUBYPATH"] = env_rubypath : ENV.delete("RUBYPATH")
@@ -147,12 +151,18 @@ class TestRequire < Test::Unit::TestCase
       t.puts "p :ok"
       t.close

-      ENV["RUBYPATH"] = "~"
-      ENV["HOME"] = t.path
-      assert_in_out_err(%w(-S test_ruby_test_require), "", [], /\(LoadError\)/)
+      env = {
+        "RUBYPATH" => "~",
+        "HOME" => t.path,
+      }
+      assert_in_out_err([env] + %w(-S test_ruby_test_require), "", [], /\(LoadError\)/)

-      ENV["HOME"], name = File.split(t.path)
-      assert_in_out_err(["-S", name], "", %w(:ok), [])
+      home, name = File.split(t.path)
+      env = {
```

```
+        "RUBYPATH" => "~",
+        "HOME" => home,
+      }
+      assert_in_out_err([env, "-S", name], "", %w(:ok), [])
     }
   ensure
     env_rubypath ? ENV["RUBYPATH"] = env_rubypath : ENV.delete("RUBYPATH")
diff --git a/test/ruby/test_rubyoptions.rb b/test/ruby/test_rubyoptions.rb
index d4e5b2a..ba62969 100644
--- a/test/ruby/test_rubyoptions.rb
+++ b/test/ruby/test_rubyoptions.rb
@@ -242,20 +242,20 @@ class TestRubyOptions < Test::Unit::TestCase
   def test_rubyopt
     rubyopt_orig = ENV['RUBYOPT']

-    ENV['RUBYOPT'] = ' - -'
-    assert_in_out_err([], "", [], [])
+    env = {'RUBYOPT' => ' - -'}
+    assert_in_out_err([env], "", [], [])

-    ENV['RUBYOPT'] = '-e "p 1"'
-    assert_in_out_err([], "", [], /invalid switch in RUBYOPT: -e \(RuntimeError\)/)
+    env['RUBYOPT'] = '-e "p 1"'
+    assert_in_out_err([env], "", [], /invalid switch in RUBYOPT: -e \(RuntimeError\)/)

-    ENV['RUBYOPT'] = '-T1'
-    assert_in_out_err(["--disable-gems"], "", [], /no program input from stdin allowed in tainted mode \(Secu
rityError\)/)
+    env['RUBYOPT'] = '-T1'
+    assert_in_out_err([env, "--disable-gems"], "", [], /no program input from stdin allowed in tainted mode \
(SecurityError\)/)

-    ENV['RUBYOPT'] = '-T4'
-    assert_in_out_err(["--disable-gems"], "", [], /no program input from stdin allowed in tainted mode \(Secu
rityError\)/)
+    env['RUBYOPT'] = '-T4'
+    assert_in_out_err([env, "--disable-gems"], "", [], /no program input from stdin allowed in tainted mode \
(SecurityError\)/)

-    ENV['RUBYOPT'] = '-Eus-ascii -KN'
-    assert_in_out_err(%w(-Eutf-8 -KU), "p '\u3042'") do |r, e|
+    env['RUBYOPT'] = '-Eus-ascii -KN'
+    assert_in_out_err([env]+%w(-Eutf-8 -KU), "p '\u3042'") do |r, e|
       assert_equal("\"\u3042\"", r.join.force_encoding(Encoding::UTF_8))
       assert_equal([], e)
     end
@@ -279,13 +279,13 @@ class TestRubyOptions < Test::Unit::TestCase
       @verbose = $VERBOSE
       $VERBOSE = nil

-      ENV['PATH'] = File.dirname(t.path)
+      env={'PATH' => File.dirname(t.path)}

-      assert_in_out_err(%w(-S) + [File.basename(t.path)], "", %w(1), [])
+      assert_in_out_err([env] + %w(-S) + [File.basename(t.path)], "", %w(1), [])

-      ENV['RUBYPATH'] = File.dirname(t.path)
+      env['RUBYPATH'] = File.dirname(t.path)

-      assert_in_out_err(%w(-S) + [File.basename(t.path)], "", %w(1), [])
+      assert_in_out_err([env] + %w(-S) + [File.basename(t.path)], "", %w(1), [])
     }

   ensure
diff --git a/test/rubygems/test_gem.rb b/test/rubygems/test_gem.rb
index 0da28bb..85556ae 100644
--- a/test/rubygems/test_gem.rb
+++ b/test/rubygems/test_gem.rb
@@ -1260,10 +1260,7 @@ class TestGem < Gem::TestCase
     install_gem b, :install_dir => path
     install_gem c, :install_dir => path

-    ENV['GEM_PATH'] = path
-    ENV['RUBYGEMS_GEMDEPS'] = "-"
-
```

```
-    out = `#{Gem.ruby.dup.untaint} -I #{LIB_PATH.untaint} -rubygems -e "p Gem.loaded_specs.values.map(&:full_
name).sort"`
+    out = `GEM_PATH=#{path} RUBYGEMS_GEMDEPS=- #{Gem.ruby.dup.untaint} -I #{LIB_PATH.untaint} -rubygems -e "p
 Gem.loaded_specs.values.map(&:full_name).sort"`

     assert_equal '["a-1", "b-1", "c-1"]', out.strip
   end
@@ -1290,12 +1287,9 @@ class TestGem < Gem::TestCase
     install_gem b, :install_dir => path
     install_gem c, :install_dir => path

-    ENV['GEM_PATH'] = path
-    ENV['RUBYGEMS_GEMDEPS'] = "-"
-
     Dir.mkdir "sub1"
     out = Dir.chdir "sub1" do
-      `#{Gem.ruby.dup.untaint} -I #{LIB_PATH.untaint} -rubygems -e "p Gem.loaded_specs.values.map(&:full_name
).sort"`
+      `GEM_PATH=#{path} RUBYGEMS_GEMDEPS=- #{Gem.ruby.dup.untaint} -I #{LIB_PATH.untaint} -rubygems -e "p Gem
.loaded_specs.values.map(&:full_name).sort"`
     end

     Dir.rmdir "sub1"
diff --git a/tool/runruby.rb b/tool/runruby.rb
index 86b9327..cf63055 100755
--- a/tool/runruby.rb
+++ b/tool/runruby.rb
@@ -66,6 +66,7 @@ config["bindir"] = abs_archdir

 env = {}

+env["HOME"] = ENV["HOME"] # RDoc relies on it
 env["RUBY"] = File.expand_path(ruby)
 env["PATH"] = [abs_archdir, ENV["PATH"]].compact.join(File::PATH_SEPARATOR)

@@ -84,8 +85,6 @@ if File.file?(libruby_so)
   end
 end

-ENV.update env
-
 cmd = [ruby]
 cmd.concat(ARGV)
 cmd.unshift(*precommand) unless precommand.empty?
@@ -96,4 +95,4 @@ if show
   puts Shellwords.join(cmd)
 end

-exec(*cmd)
+exec(env, *cmd)
--
1.9.1
```

#8 - 10/18/2014 10:38 PM - jeremyevans0 (Jeremy Evans)

I agree with Eric, this change is too likely to break things, including the principle of least surprise. No other program language I'm aware of spawns system commands with an empty environment by default. Here are a few I tried:

awk 'BEGIN {system("env")}'
perl -e "system('env')"
ruby -e "system('env')"
python -c 'import os; os.system("env")'
php -r "system('env');"
lua -e 'os.execute("env")'
clojure -e "(use '[clojure.java.shell :only [sh]]) (println (:out (sh \"env\"))) (System/exit 0)"
gforth -e 'sh "env"' -e bye

The costs of this proposed change are much higher than the benefits. This just makes interaction with the operating system in general more difficult. It is likely this patch introduces more vulnerabilities than it fixes, in addition to breaking things.

#9 - 10/20/2014 04:39 PM - shyouhei (Shyouhei Urabe)

Jeremy Evans wrote:

> The costs of this proposed change are much higher than the benefits. This just makes interaction with the operating system in general more

difficult. It is likely this patch introduces more vulnerabilities than it fixes, in addition to breaking things.

I'm not a fundamentalist here. If anyone could propose a moderate way to samitize malicious environment variables, that should be taken seriously.

After shellshock we have to realize that environment variables are not as safe as we thought them to be. I don't think it's a good thing to neglect this fact to blame bash only. I'm proposing what I can do.

#10 - 10/20/2014 08:29 PM - normalperson (Eric Wong)

How about supporting ENV.freeze instead?
Currently ENV.freeze is a no-op, this patch changes that:

http://80x24.org/spew/m/400e216159e74b65608f2f0b296817cc9823e3bb.txt

```
ENV.freeze should behave like Hash#freeze and not allow future
modifications to the ENV from pure Ruby libraries.

Users may call ENV.freeze to prevent 3rd-party (pure) Ruby libraries
from modifying the process environment any further.

This cannot not defend against users who modify the environment using
3rd-party C extensions, Fiddle, or FFI RubyGem
```

#11 - 10/22/2014 09:11 AM - shyouhei (Shyouhei Urabe)

I see this patch is (basically) good to have.

One thing sprung in my mind is that perhaps we should also freeze each environment variables, not only the ENV object. That is not the same way Hash#freeze works, but ENV keys are already frozen as-is so also freezing values might be an opiton. That should prevent modifying already-existent environment variables, such as RUBYOPT.

On 10/21/2014 05:24 AM, Eric Wong wrote:

> How about supporting ENV.freeze instead?
> Currently ENV.freeze is a no-op, this patch changes that:
>
> http://80x24.org/spew/m/400e216159e74b65608f2f0b296817cc9823e3bb.txt
>
> ```
> ENV.freeze should behave like Hash#freeze and not allow future
> modifications to the ENV from pure Ruby libraries.
>
> Users may call ENV.freeze to prevent 3rd-party (pure) Ruby libraries
> from modifying the process environment any further.
>
> This cannot not defend against users who modify the environment using
> 3rd-party C extensions, Fiddle, or FFI RubyGem
> ```

#12 - 01/05/2018 09:01 PM - naruse (Yui NARUSE)

- Target version deleted (2.2.0)