Ruby 1.8 - Bug #5105

CGI::Session#session_id の生成方法について

07/27/2011 12:48 PM - tommy (Masahiro Tomita)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	
ruby -v:	-

Description

とみたです。

古い話ですが、r13672 で CGI::Session#session_id が SecureRandom が生成した乱数を単純に使用するようになっています。それまではタイムスタンプ & プロセスID & 乱数 & 固定文字から生成した MD5 ダイジェスト値が使用されていました。

このように生成した MD5 ダイジェスト値よりも、単純に乱数をそのまま使用する方が重複が発生しやすくなってしまっているんじゃないかと思うのですがどうでしょう。

というか、実際に重複が発生してしまったので。

以下、SecureRandom を使いつつ元の挙動に戻すパッチです。

```
--- lib/cgi/session.rb.orig 2009-02-20 19:35:11.000000000 +0900
+++ lib/cgi/session.rb  2011-07-27 12:27:57.000000000 +0900
@@ -25,6 +25,8 @@

require 'cgi'
require 'tmpdir'
+require 'securerandom'
+require 'digest/md5'

class CGI

@@ -174,21 +176,15 @@
# is used internally for automatically generated
# session ids.
def create_new_id

  • require 'securerandom'
  • begin
  • session_id = SecureRandom.hex(16)
  • rescue NotImplementedError
  • require 'digest/md5'
  • md5 = Digest::MD5::new
  • now = Time::now
  • md5.update(now.to_s)
  • md5.update(String(now.usec))
  • md5.update(String(rand(0)))
  • md5.update(String($$))
  • md5.update('foobar')
  • session_id = md5.hexdigest
  • end
  • session_id
  • r = SecureRandom.random_bytes(16) rescue rand(0).to_s
  • md5 = Digest::MD5::new
  • now = Time::now
  • md5.update(now.to_s)
  • md5.update(String(now.usec))
  • md5.update(r)
```

- md5.update(String($$))
- md5.update('foobar')
- md5.hexdigest end private :create_new_id

History

#1 - 07/27/2011 04:40 PM - naruse (Yui NARUSE)

まともな乱数・まともなハッシュ関数を前提とした場合、衝突確率は単純に空間の広さに依存するので、
16バイトならば空間はMD5と等しく、重複する確率は同じはずです。

単純に運が悪かったと言えるような確率でもないので、何かを踏んだ可能性が高い気はしますが

#2 - 07/27/2011 07:53 PM - tommy (Masahiro Tomita)

とみたです。

On Wed, 27 Jul 2011 16:40:18 +0900
Yui NARUSE naruse@airemix.jp wrote:

> まともな乱数・まともなハッシュ関数を前提とした場合、衝突確率は単純に空間の広さに依存するので、
> 16バイトならば空間はMD5と等しく、重複する確率は同じはずです。

なるほど。

> 単純に運が悪かったと言えるような確率でもないので、何かを踏んだ可能性が高い気はしますが

ですよねぇ...。最初にそう思って色々と調べてみたのですが...。
もうちょっと考えてみます。

--
とみたまさひろ tommy@tmtm.org
http://twitter.com/tmtms
D68F 8F55 7F6C 5908 88EB 1EBA 25ED DEE7 BBE8 1752

#3 - 07/27/2011 07:53 PM - tommy (Masahiro Tomita)

とみたです。

On Wed, 27 Jul 2011 16:40:18 +0900
Yui NARUSE naruse@airemix.jp wrote:

> まともな乱数・まともなハッシュ関数を前提とした場合、衝突確率は単純に空間の広さに依存するので、
> 16バイトならば空間はMD5と等しく、重複する確率は同じはずです。

なるほど。

> 単純に運が悪かったと言えるような確率でもないので、何かを踏んだ可能性が高い気はしますが

ですよねぇ...。最初にそう思って色々と調べてみたのですが...。
もうちょっと考えてみます。

--
とみたまさひろ tommy@tmtm.org
http://twitter.com/tmtms
D68F 8F55 7F6C 5908 88EB 1EBA 25ED DEE7 BBE8 1752

#4 - 07/28/2011 12:53 AM - naruse (Yui NARUSE)

- ruby -v changed from ruby 1.8.7 (2011-06-30 patchlevel 352) [i686-linux] to -

(2011/07/27 19:47), とみたまさひろ wrote:

> On Wed, 27 Jul 2011 16:40:18 +0900
> Yui NARUSE naruse@airemix.jp wrote:
>
>> まともな乱数・まともなハッシュ関数を前提とした場合、衝突確率は単純に空間の広さに依存するので、
>> 16バイトならば空間はMD5と等しく、重複する確率は同じはずです。

> なるほど。
>
>> 単純に運が悪かったと言えるような確率でもないので、何かを踏んだ可能性が高い気はしますが
>
> ですよねぇ…。最初にそう思って色々と調べてみたのですが…。
> もうちょっと考えてみます。

あ、わかった、r32050 ですね。
つまり、Ruby 1.8.7 ならば最新のパッチレベルを使って頂ければ大丈夫なはずです。

--
NARUSE, Yui naruse@airemix.jp

#5 - 07/28/2011 12:53 AM - naruse (Yui NARUSE)

(2011/07/27 19:47), とみたまさひろ wrote:

> On Wed, 27 Jul 2011 16:40:18 +0900
> Yui NARUSE naruse@airemix.jp wrote:
>
>> まともな乱数・まともなハッシュ関数を前提とした場合、衝突確率は単純に空間の広さに依存するので、
>> 16バイトならば空間はMD5と等しく、重複する確率は同じはずです。
>
> なるほど。
>
>> 単純に運が悪かったと言えるような確率でもないので、何かを踏んだ可能性が高い気はしますが
>
> ですよねぇ…。最初にそう思って色々と調べてみたのですが…。
> もうちょっと考えてみます。

あ、わかった、r32050 ですね。
つまり、Ruby 1.8.7 ならば最新のパッチレベルを使って頂ければ大丈夫なはずです。

--
NARUSE, Yui naruse@airemix.jp

#6 - 07/28/2011 02:23 AM - tommy (Masahiro Tomita)

とみたです。

On Thu, 28 Jul 2011 00:40:28 +0900
"NARUSE, Yui" naruse@airemix.jp wrote:

> あ、わかった、r32050 ですね。
> つまり、Ruby 1.8.7 ならば最新のパッチレベルを使って頂ければ大丈夫なはずです。

おお! ありがとうございます!

というか、1.8.7-p352 を使ってたつもりが、現象が発生した環境は
1.8.7-p334 でした… orz...

こんな感じ↓のワンライナーで、同じPIDの時に同じ乱数が生成されることと、
p352 で直ってることが確認できました。

```
% ruby -rsecurerandom -e 'p [$$, SecureRandom.hex(16)]; 33000.times{pid=fork{p [$$,SecureRandom.hex(16)]}; Process.waitpid pid}'
```

ところで、色々と調べている時に見つけたのですが、securerandom.rb の次の行で、

```
     ary = [now.to_i, now.usec, @pid, pid]
     OpenSSL::Random.seed(ary.to_s)
```

1秒以内に特定の usec と PID のパターンが発生すると、1.8.7 では同じ
ary.to_s が生成されるように思うのですが、問題ないでしょうか。たとえば、

now.to_i	now.usec	@pid	pid
1311785953	11	2222	22233
1311785953	1122	2222	233

この例だと、ary.to_s の結果が両方とも "13117859531122222233" になります。

1.9.2 だと Array#to_s の表現が変わったので同じにはなりませんでした。

ary.to_s じゃなくて ary.join('.') 等のようにした方がいいんじゃないかと
思います。

--
とみたまさひろ tommy@tmtm.org
http://twitter.com/tmtms
D68F 8F55 7F6C 5908 88EB 1EBA 25ED DEE7 BBE8 1752

#7 - 07/28/2011 02:23 AM - tommy (Masahiro Tomita)

とみたです。

On Thu, 28 Jul 2011 00:40:28 +0900
"NARUSE, Yui" naruse@airemix.jp wrote:

> あ、わかった、r32050 ですね。
> つまり、Ruby 1.8.7 ならば最新のパッチレベルを使って頂ければ大丈夫なはずです。

おお！ ありがとうございます！

というか、1.8.7-p352 を使ってたつもりが、現象が発生した環境は
1.8.7-p334 でした… orz...

こんな感じ↓のワンライナーで、同じPIDの時に同じ乱数が生成されることと、
p352 で直ってることが確認できました。

```
% ruby -rsecurerandom -e 'p [$$, SecureRandom.hex(16)]; 33000.times{pid=fork{p [$$,SecureRandom.hex(16)]}; Pro
cess.waitpid pid}'
```

ところで、色々と調べている時に見つけたのですが、securerandom.rb の次の行で、

```
      ary = [now.to_i, now.usec, @pid, pid]
      OpenSSL::Random.seed(ary.to_s)
```

1秒以内に特定の usec と PID のパターンが発生すると、1.8.7 では同じ
ary.to_s が生成されるように思うのですが、問題ないでしょうか。たとえば、

now.to_i	now.usec	@pid	pid
1311785953	11	2222	22233
1311785953	1122	2222	233

この例だと、ary.to_s の結果が両方とも "13117859531122222233" になります。

1.9.2 だと Array#to_s の表現が変わったので同じにはなりませんでした。

ary.to_s じゃなくて ary.join('.') 等のようにした方がいいんじゃないかと
思います。

--
とみたまさひろ tommy@tmtm.org
http://twitter.com/tmtms
D68F 8F55 7F6C 5908 88EB 1EBA 25ED DEE7 BBE8 1752

#8 - 07/29/2011 08:23 PM - tommy (Masahiro Tomita)

とみたです。

On Thu, 28 Jul 2011 02:13:07 +0900
とみたまさひろ tommy@tmtm.org wrote:

> こんな感じ↓のワンライナーで、同じPIDの時に同じ乱数が生成されることと、
> p352 で直ってることが確認できました。
>
> % ruby -rsecurerandom -e 'p [$$, SecureRandom.hex(16)]; 33000.times{pid=fork{p [$$,SecureRandom.hex(16)]}; Process.waitpid pid}'

ダメでした…。次のスクリプトで同じ乱数が生成されました。

```
% ruby -rsecurerandom -e 'OpenSSL::Random.random_bytes(16); 33000.times{pid=fork{p [$$,SecureRandom.hex(16)]}; Process.waitpid pid}'
```

SecureRandom.random_bytes がそのプロセスで最初に呼ばれた時には常に
OpenSSL::Random.seed を呼ぶようにしてみたらうまくいきました。先のメール
の ary.to_s も一緒に修正しています。

Index: lib/securerandom.rb
===
--- lib/securerandom.rb   (revision 32735)
+++ lib/securerandom.rb   (working copy)
@@ -50,12 +50,12 @@
def self.random_bytes(n=nil)
n ||= 16
if defined? OpenSSL::Random

- @pid = $$ if !defined?(@pid)
- @pid = 0 if !defined?(@pid)   pid = $$ if @pid != pid   now = Time.now   ary = [now.to_i, now.usec, @pid, pid]
- OpenSSL::Random.seed(ary.to_s)
- OpenSSL::Random.seed(ary.join('.'))   @pid = pid   end   return OpenSSL::Random.random_bytes(n)

--
とみたまさひろ tommy@tmtm.org
http://twitter.com/tmtms
D68F 8F55 7F6C 5908 88EB 1EBA 25ED DEE7 BBE8 1752

#9 - 07/29/2011 08:23 PM - tommy (Masahiro Tomita)

とみたです。

On Thu, 28 Jul 2011 02:13:07 +0900
とみたまさひろ tommy@tmtm.org wrote:

> こんな感じ↓のワンライナーで、同じPIDの時に同じ乱数が生成されることと、
> p352 で直ってることが確認できました。
>
> % ruby -rsecurerandom -e 'p [$$, SecureRandom.hex(16)]; 33000.times{pid=fork{p [$$,SecureRandom.hex(16)]}; Process.waitpid pid}'

ダメでした...。次のスクリプトで同じ乱数が生成されました。

% ruby -rsecurerandom -e 'OpenSSL::Random.random_bytes(16); 33000.times{pid=fork{p [$$,SecureRandom.hex(16)]}; Process.waitpid pid}'

SecureRandom.random_bytes がそのプロセスで最初に呼ばれた時には常に
OpenSSL::Random.seed を呼ぶようにしてみたらうまくいきました。先のメール
の ary.to_s も一緒に修正しています。

Index: lib/securerandom.rb
===
--- lib/securerandom.rb   (revision 32735)
+++ lib/securerandom.rb   (working copy)
@@ -50,12 +50,12 @@
def self.random_bytes(n=nil)
n ||= 16
if defined? OpenSSL::Random

- @pid = $$ if !defined?(@pid)
- @pid = 0 if !defined?(@pid)   pid = $$ if @pid != pid   now = Time.now   ary = [now.to_i, now.usec, @pid, pid]
- OpenSSL::Random.seed(ary.to_s)
- OpenSSL::Random.seed(ary.join('.'))   @pid = pid   end   return OpenSSL::Random.random_bytes(n)

--
とみたまさひろ tommy@tmtm.org
http://twitter.com/tmtms
D68F 8F55 7F6C 5908 88EB 1EBA 25ED DEE7 BBE8 1752

#10 - 07/29/2011 11:59 PM - akr (Akira Tanaka)

2011年7月29日20:04 とみたまさひろ tommy@tmtm.org:

> ダメでした...。次のスクリプトで同じ乱数が生成されました。
>
> % ruby -rsecurerandom -e 'OpenSSL::Random.random_bytes(16); 33000.times{pid=fork{p [$$,SecureRandom.hex(16)]}; Process.waitpid pid}'

なるほど。securerandom.rb 以外で OpenSSL::Random.random_bytes が呼ばれると、
securerandom.rb は fork したことに気がつけないのですね。

> SecureRandom.random_bytes がそのプロセスで最初に呼ばれた時には常に
> OpenSSL::Random.seed を呼ぶようにしてみたらうまくいきました。先のメール
> の ary.to_s も一緒に修正しています。

入れておきます。
--
[田中 哲][たなか あきら][Tanaka Akira]

#11 - 07/29/2011 11:59 PM - akr (Akira Tanaka)

2011年7月29日20:04 とみたまさひろ tommy@tmtm.org:

> ダメでした...。次のスクリプトで同じ乱数が生成されました。
>
> ```
> % ruby -rsecurerandom -e 'OpenSSL::Random.random_bytes(16); 33000.times{pid=fork{p [$$,SecureRandom.hex(16)]}; Process.waitpid pid}'
> ```

なるほど。securerandom.rb 以外で OpenSSL::Random.random_bytes が呼ばれると、
securerandom.rb は fork したことに気がつけないのですね。

> SecureRandom.random_bytes がそのプロセスで最初に呼ばれた時には常に
> OpenSSL::Random.seed を呼ぶようにしてみたらうまくいきました。先のメール
> の ary.to_s も一緒に修正しています。

入れておきます。
--
[田中 哲][たなか あきら][Tanaka Akira]

#12 - 08/13/2011 05:53 PM - naruse (Yui NARUSE)

(2011/07/29 23:55), Tanaka Akira wrote:

> 2011年7月29日20:04 とみたまさひろ tommy@tmtm.org:
>
> > ダメでした...。次のスクリプトで同じ乱数が生成されました。
> >
> > ```
> > % ruby -rsecurerandom -e 'OpenSSL::Random.random_bytes(16); 33000.times{pid=fork{p [$$,SecureRandom.hex(16)]}; Process.waitpid pid}'
> > ```
>
> なるほど。securerandom.rb 以外で OpenSSL::Random.random_bytes が呼ばれると、
> securerandom.rb は fork したことに気がつけないのですね。
>
> > SecureRandom.random_bytes がそのプロセスで最初に呼ばれた時には常に
> > OpenSSL::Random.seed を呼ぶようにしてみたらうまくいきました。先のメール
> > の ary.to_s も一緒に修正しています。
>
> 入れておきます。

ふと思ったのですが、 openssl を使うときでも、/dev/urandom があるならば、
openssl の seed に /dev/urandom を一部使ってもいいんじゃないでしょうか。

--
NARUSE, Yui naruse@airemix.jp

#13 - 08/13/2011 07:23 PM - akr (Akira Tanaka)

2011年8月13日17:31 NARUSE, Yui naruse@airemix.jp:

> ふと思ったのですが、 openssl を使うときでも、/dev/urandom があるならば、
> openssl の seed に /dev/urandom を一部使ってもいいんじゃないでしょうか。

fork する頻度で /dev/urandom から読み出すことが望ましいか、
という問題ですかね。
--
[田中 哲][たなか あきら][Tanaka Akira]

#14 - 08/13/2011 08:53 PM - kosaki (Motohiro KOSAKI)

> > ふと思ったのですが、 openssl を使うときでも、/dev/urandom があるならば、
> > openssl の seed に /dev/urandom を一部使ってもいいんじゃないでしょうか。
>
> fork する頻度で /dev/urandom から読み出すことが望ましいか、
> という問題ですかね。

あんまり詳しくないんですが。

PRNGである以上、seedが予測されたら、全乱数列が予測されてしまうので、
安全側に倒すほうがエントロピー枯渇よりも重要なんじゃないかという気がするんですけどねえ・・

OSのレベルでも、ASLRの実現のためにprocess
spawnするたびに/dev/urandom読んでるので、許容範囲内ではないかなあ。恣意的な運用を考えたらダメなケースは発生しうるかもしれないけど

#15 - 08/13/2011 11:23 PM - naruse (Yui NARUSE)

(2011/08/13 20:35), KOSAKI Motohiro wrote:

> > > ふと思ったのですが、 openssl を使うときでも、/dev/urandom があるならば、
> > > openssl の seed に /dev/urandom を一部使ってもいいんじゃないでしょうか。
> >
> > fork する頻度で /dev/urandom から読み出すことが望ましいか、
> > という問題ですかね。
>
> あんまり詳しくないんですが。
>
> PRNGである以上、seedが予測されたら、全乱数列が予測されてしまうので、
> 安全側に倒すほうがエントロピー枯渇よりも重要なんじゃないかという気がするんですけどねえ・・
>
> OSのレベルでも、ASLRの実現のためにprocess
> spawnするたびに/dev/urandom読んでるので、許容範囲内ではないかなあ。恣意的な運用を考えたらダメなケースは発生しうるかもしれないけど

/dev/random ではなく urandom なのでいいんじゃないでしょうか、
もちろんエントロピーは消費するわけですが。

加えて、現在攻撃側に経って考えると usec/nsec * pid あたりが予測不可能性を担保しているところ、
これって 100万~1億 * 6万なわけで、42bit くらいしか空間ありませんよね。
ちょっと seed が弱いと思うんです。

--
NARUSE, Yui naruse@airemix.jp

#16 - 08/14/2011 05:53 AM - akr (Akira Tanaka)

2011年8月13日23:12 NARUSE, Yui naruse@airemix.jp:

> > PRNGである以上、seedが予測されたら、全乱数列が予測されてしまうので、
> > 安全側に倒すほうがエントロピー枯渇よりも重要なんじゃないかという気がするんですけどねえ・・
> >
> > OSのレベルでも、ASLRの実現のためにprocess
> > spawnするたびに/dev/urandom読んでるので、許容範囲内ではないかなあ。恣意的な運用を考えたらダメなケースは発生しうるかもしれないけど
>
> /dev/random ではなく urandom なのでいいんじゃないでしょうか、
> もちろんエントロピーは消費するわけですが。

forking server だと request ごとに消費しちゃうというのが嫌な感じなのですが、
OS 自体の消費に対して線形ならいいかなぁと思いつつも、

> 加えて、現在攻撃側に経って考えると usec/nsec * pid あたりが予測不可能性を担保しているところ、
> これって 100万~1億 * 6万なわけで、42bit くらいしか空間ありませんよね。
> ちょっと seed が弱いと思うんです。

これは違います。
RAND_seed() はその時点の種にデータを混ぜるものなので、
最初に openssl が /dev/urandom から読んだ種が消えるわけではありません。

RAND_add(3SSL):

RAND_add() mixes the num bytes at buf into the PRNG state.
--
[田中 哲][たなか あきら][Tanaka Akira]

#17 - 08/15/2011 12:59 PM - Anonymous

> forking server だと request ごとに消費しちゃうというのが嫌な感じなのですが、
> OS 自体の消費に対して線形ならいいかなぁと思いつつも、
>
>> 加えて、現在攻撃側に経って考えると usec/nsec * pid あたりが予測不可能性を担保しているところ、
>> これって 100万~1億 * 6万なわけで、42bit くらいしか空間ありませんよね。
>> ちょっと seed が弱いと思うんです。
>
> これは違います。
> RAND_seed() はその時点の種にデータを混ぜるものなので、
> 最初に openssl が /dev/urandom から読んだ種が消えるわけではありません。
>
> RAND_add(3SSL):
> RAND_add() mixes the num bytes at buf into the PRNG state.

おっと。僕はここを誤解していました。
なので先の主張は撤回したいと思います。

#18 - 08/15/2011 05:59 PM - naruse (Yui NARUSE)

2011年8月15日12:55 KOSAKI Motohiro kosaki.motohiro@jp.fujitsu.com:

>> forking server だと request ごとに消費しちゃうというのが嫌な感じなのですが、
>> OS 自体の消費に対して線形ならいいかなぁと思いつつも、
>>
>>> 加えて、現在攻撃側に経って考えると usec/nsec * pid あたりが予測不可能性を担保しているところ、
>>> これって 100万~1億 * 6万なわけで、42bit くらいしか空間ありませんよね。
>>> ちょっと seed が弱いと思うんです。
>>
>> これは違います。
>> RAND_seed() はその時点の種にデータを混ぜるものなので、
>> 最初に openssl が /dev/urandom から読んだ種が消えるわけではありません。
>>
>> RAND_add(3SSL):
>> RAND_add() mixes the num bytes at buf into the PRNG state.
>
> おっと。僕はここを誤解していました。
> なので先の主張は撤回したいと思います。

わたしも誤解していました。
しばらく考えたんですが前出のパッチ適用後は攻撃できる気がしないので撤回します。

--
NARUSE, Yui naruse@airemix.jp