NUMBER 3 — JANUARY 25, 1996

RSA Laboratories' Bulletin

News and advice from RSA Laboratories

Proper Initialization for the BSAFE Random Number Generator

Dr. Robert W. Baldwin

RSA Data Security, Inc.

Abstract

This bulletin will help you avoid a particularly poor way of initializing the random number generators in the BSAFE 1.x or BSAFE 2.x toolkit. If you have followed the advice in the BSAFE 2.1 User's Manual, which is to initialize the generators with large blocks of seed bytes, then you do not have this security weakness. However, random number generators are often used to produce encryption keys, so the impact of poor initialization could be substantial.

We first explain how you can recognize whether your product has this problem. If it does, we have included a simple fix that can be added to your product's source code. Alternatively, you can upgrade to the BSAFE 3.x toolkit which eliminates the problem without any need for changes in your product's source code.

Recognizing the problem

You need to look at all places that your product calls B_RandomUpdate. If the number of seed bytes passed to each invocation of B_RandomUpdate is small, then you may have the weakness, even if B_RandomUpdate is called may times.

In the sample code, B_RandomUpdate is called 20 times, but each call only passes a single seed byte. The algorithm used to update the state of the random number generator (in BSAFE 1.x and BSAFE 2.x) assumes that each call will pass a substantial amount of unpredictability in each block of seed bytes. However, in this case, there is only one byte in each seed block, and it is likely to be a character from the home row of the keyboard, of which there are only 10. That is very little unpredictability per seed block.

```
DisplayMessage
   (“Please enter 20 random keystrokes”);
for (i = 0 ; i < 20 ; i++)
{
   c = GetUserKeystroke();
   if (0 != B_RandomUpdate(randomObj,
            &c, 1, NULL_CONTEXT))
       return (ERROR_XXX);
}
```

The number of states for the generator does not depend on the order in which the seed blocks are processed by B_RandomUpdate. After executing the sample code, the random number generator will be in one of (20+10-1)!/(20!(10-1)!) possible states, which is about 23 bits of randomness. This is not even enough randomness to generate a good 40 bit key for an exportable cryptographic product, let alone provide strong domestic security. The implementer of this code may have expected that there would be 10**20 possible states, roughly 66 bits of randomness, which is reasonable for systems that use the DES cipher.

Fixing the problem

There are several ways to fix the problem. One would be to gather up all the seed bytes and pass

Bob Baldwin is a senior engineer at RSA Data Security. He can be contacted via baldwin@rsa.com.

them via a single call to B_RandomUpdate. This would extract the maximum amount of randomness from all the seed bytes. For example:

```
DisplayMessage
   (“Please enter 20 random keystrokes”);
for (i = 0 ; i < 20 ; i++)
{
   seedBuffer[i] = GetUserKeystroke();
}
if (0 != B_RandomUpdate(randomObj,
         seedBuffer, 20, NULL_CONTEXT))
    return (ERROR_XXX);
```

An alternative is to include a counter value with each byte of seed. Surprisingly, this fixes the problem with small seeds. For user input, the counter could be replaced with a sample from a clock that is updated at least 10 times per second. The code for this alternative is shown below. This approach extracts nearly as much randomness from the input as the previous approach and it is well suited for applications that periodically update the random object as they executes.

```
/* Update the random object anytime a key is pressed. */
int AddNewRandomness ()
{
    /* Using a clock (10th of second OK) would be better than */
    /* a counter and would not require static storage. */
    static struct {  /* Static, so counter is persistent. */
        long    counter;
        char    keystroke;
    } seedBlock;

    seedBlock.counter++;
    seedBlock.keystroke = GetLastKeystroke();
    if (0 != B_RandomUpdate(randomObj,
                           (char *)&seedBlock,sizeof(seedBlock),
                            NULL_CONTEXT))
        return (ERROR_XXX);
    return (0);
}
```

For further information contact technical support by calling (415)595-7705 between 9 A.M. and 5 P.M. Pacific time, or fax at (415)595-1873, or email at *tech-support@rsa.com*.

For more information on this and other recent developments in cryptography, contact RSA Laboratories at one of the addresses below.

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065 USA
415/595-7703
415/595-4126 (fax)
rsa-labs@rsa.com
http://www.rsa.com/rsalabs/