

# Engineering MD5 Collisions in Executable Binaries

Shu Ning Bian

November 26, 2005

## Abstract

With the recent release of an efficient MD5 collision generator by Stach & Liu, it is now feasible to engineer executable binaries to have the same MD5 sum but different behaviour. One such technique is presented in this paper, along with all relevant source code and documentation. The method described here is simple and applicable to a wide range of build and execution environments, and executable formats. This technique was employed on an x86-32 platform running SuSE 10.0.

## 1 Introduction

The MD5 algorithm contains a well-known weakness whereby if two prefixes have the same MD5 hash, appending a common suffix will also result in a MD5 collision. Previously it has not been feasible to artificially produce MD5 collisions on demand, so this weakness remained academic. However, recent developments have allowed random binary strings which cause MD5 collisions to be generated easily with commodity hardware. It has thus become possible to engineer executable binaries with identical MD5 sums. First, a program is constructed that behaves differently depending on the equality of two 1024-bit arrays, which are contained in the program. This program is then compiled in order to determine the file offset at which the first array is located. Using this offset, the state of the MD5 digest of the binaries up to that point is observed. The state values then form the initialisation vector parameters of `md5coll[2]`, which produces two binary strings with the same MD5 sum. These binary strings are then placed in the aforementioned arrays in such a way that an MD5 collision occurs, along with behavioural modification. A log of how each of these steps was performed can be found in the Appendix.

It should be noted that this technique is *not a new attack vector* for security exploits, as the requirements for the use of this procedure far exceed other more subtle and effective methods.

## 2 The Programs

The programs modified to give the same MD5 sum may be as simple or as complicated as desired; even existing programs may be successfully used. In any case, two 1024-bit arrays, `m0` and `m1`, are required. A simple loop then checks the equality of the arrays and the program behaviour is determined by the result of this comparison. This code may be located anywhere - it does not need to be placed at the beginning of the binary files, or in any particular function. The important thing to note is that the programs must not use any variables which depend on the contents of `m0` and `m1`, as any such variables will change with `md5coll`'s output, rendering the results invalid. A sample implementation of two programs with these requirements is included in the Appendix.

## 3 MD5 State Discovery

After compilation of the program in the previous section, we locate the file offset where the array `m0` occurs. This is a simple task with the aid of a hex editor. Using a modified RSA MD5 Message Digest Algorithm implementation[1], the state of the MD5 digest of the binaries at the beginning of `m0` is then determined. This is represented by four variables,  $IV_1 \dots IV_4$ . The listing of the modified MD5 implementation can be found in the Appendix.

## 4 Finding Binary Strings with MD5 Collision

This step is the heart of the technique. Stach & Liu's publicly released `md5coll.c` was compiled and run with the command line arguments  $IV_1$ ,  $IV_2$ ,  $IV_3$ , and  $IV_4$ . Upon completion, `md5coll` generates C declarations for two arrays of 1024 bits each, which produces an MD5 collision. If the program to be modified is written in C, then these declarations can be used as given. Otherwise language-dependent conversions are necessary. Endian differences must also be taken into account.

## 5 Array Substitution and Compilation

The `md5coll` program gives the arrays `m0` and `m1`. Suppose now the programs constructed previously are named `good` and `bad`, where `good` behaviour occurs when the arrays are equal, and `bad` behaviour otherwise. `m1` should be placed in the first array of `good`, and `m0` in the first array of `bad`. To ensure the suffices are equal, the second arrays in both `good` and `bad` need to be identical. From the requirements for `good` and `bad`, the second array must be equal to `m1` to trigger the desired behaviour. Recompiling the programs now yields two executable binaries that have the same MD5 sum, but different behaviour.

## 6 Results

Below is the output resulting from an implementation of this technique:

```
steve@rei:~/code/md5coll> md5sum good2 bad2
1235f679c2250f08e750e45e0eabe1f2  good2
1235f679c2250f08e750e45e0eabe1f2  bad2
steve@rei:~/code/md5coll> ls -l good2 bad2
-rwxr-xr-x  1 steve users 3648 2005-11-17 22:36 bad2
-rwxr-xr-x  1 steve users 3648 2005-11-17 22:36 good2
steve@rei:~/code/md5coll> ./good2
Hello world!
steve@rei:~/code/md5coll> ./bad2
Die world!
steve@rei:~/code/md5coll>
```

## 7 Problems

There are several issues with this technique, the most significant being the amount of access required in order to carry out an attack. An attacker with the ability to perform this trick already has more than enough power to do harm using far simpler and more subtle methods. For example, substituting `=` for `==` in the source code. Another problem is that code for both behaviours are included in each binary, so detection of malicious code will potentially occur for both binaries. Lastly, this technique is an overly complicated way of sneaking Trojan code into a system, then activating it.

## 8 Conclusions

Release of efficient MD5 collision generation code by Stach & Liu is another nail in MD5's coffin. However until pre-imaging attacks are possible, MD5 has not seen the end of its days. There is still no viable way to inject malicious code into existing binaries whilst preserving the MD5 sums. The technique described here only allows two binaries to be engineered so that they cause a MD5 collision. All in all, this method is interesting but harmless.

## 9 Acknowledgements

In no particular order, the following people gave invaluable help in the preparation of this paper: Josh Padman, Emma Fitzgerald, and Neil Archibald.

## References

- [1] Inc RSA Data Security. RSA MD5 Message Digest Algorithm.  
<http://theory.lcs.mit.edu/~rivest/md5.c>, 1990.
- [2] Stach and Liu. MD5 Collision Generation.  
<http://www.stachliu.com/collisions.html>, 2005.

## Appendix

---

**Algorithm 1** good2.c before array substitution

---

```
#include <stdio.h>
unsigned int m0[32] = {
    0xb3ebceba, 0x153fe51d, 0xc6effbf0, 0xabe408f5,
    0x0facadfd, 0x800f6d23, 0xb6520ed1, 0x4643071f,
    0x05332cb1, 0xdf747df8, 0x76b6f147, 0x52fa035c,
    0x221c4917, 0x66da5620, 0xb9dcbb71, 0xcacc629b,
    0x192fdffb, 0x14d812b3, 0x9e36b7f4, 0x949f528c,
    0xa7225702, 0xee44622e, 0xf75cb50a, 0xb2bdd7d3,
    0x77b37fcd, 0xea48561a, 0xa77f992a, 0x2011f89a,
    0x700498b9, 0x27080d65, 0xdac762e0, 0x76da25bc,
};
unsigned int m1[32] = {
    0xb3ebceba, 0x153fe51d, 0xc6effbf0, 0xabe408f5,
    0x0facadfd, 0x800f6d23, 0xb6520ed1, 0x4643071f,
    0x05332cb1, 0xdf747df8, 0x76b6f147, 0x52fa035c,
    0x221c4917, 0x66da5620, 0xb9dcbb71, 0xcacc629b,
    0x192fdffb, 0x14d812b3, 0x9e36b7f4, 0x949f528c,
    0xa7225702, 0xee44622e, 0xf75cb50a, 0xb2bdd7d3,
    0x77b37fcd, 0xea48561a, 0xa77f992a, 0x2011f89a,
    0x700498b9, 0x27080d65, 0xdac762e0, 0x76da25bc,
};
int main(void)
{
    int i;
    for ( i = 0; i < 32; ++i)
    {
        if ( m0[i] != m1[i])
        {
            puts("Die world!");
            return 0;
        }
    }
    puts("Hello world!");
    return 0;
}
```

---

---

**Algorithm 2** bad2.c before array substitution

---

```
#include <stdio.h>
unsigned int m0[32] = {
    0xb3ebceba, 0x153fe51d, 0xc6effbf0, 0xabe408f5,
    0x0facadfd, 0x800f6d23, 0xb6520ed1, 0x4643071f,
    0x05332cb1, 0xdf747df8, 0x76b6f147, 0x52fa035c,
    0x221c4917, 0x66da5620, 0xb9dcbb71, 0xcacc629b,
    0x192fdffb, 0x14d812b3, 0x9e36b7f4, 0x949f528c,
    0xa7225702, 0xee44622e, 0xf75cb50a, 0xb2bdd7d3,
    0x77b37fcd, 0xea48561a, 0xa77f992a, 0x2011f89a,
    0x700498b9, 0x27080d65, 0xdac762e0, 0x76da25bc,
};
unsigned int m1[32] = {
    0xb3ebceba, 0x153fe51d, 0xc6effbf0, 0xabe408f5,
    0x8facadfd, 0x800f6d23, 0xb6520ed1, 0x4643071f,
    0x05332cb1, 0xdf747df8, 0x76b6f147, 0x52fa835c,
    0x221c4917, 0x66da5620, 0x39dcbb71, 0xcacc629b,
    0x192fdffb, 0x14d812b3, 0x9e36b7f4, 0x949f528c,
    0x27225702, 0xee44622e, 0xf75cb50a, 0xb2bdd7d3,
    0x77b37fcd, 0xea48561a, 0xa77f992a, 0x2011789a,
    0x700498b9, 0x27080d65, 0x5ac762e0, 0x76da25bc,
};
int main(void)
{
    int i;
    for ( i = 0; i < 32; ++i)
    {
        if ( m0[i] != m1[i])
        {
            puts("Die world!");
            return 0;
        }
    }
    puts("Hello world!");
    return 0;
}
```

---

---

**Algorithm 3** Finding the file offset of the first array and discovering MD5 state variables

---

```
steve@rei:~/code/md5coll> hexdump test10.bin
00000000 ceba b3eb e51d 153f fbf0 c6ef 08f5 abe4
00000010 adfd 0fac 6d23 800f 0ed1 b652 071f 4643
00000020 2cb1 0533 7df8 df74 f147 76b6 035c 52fa
00000030 4917 221c 5620 66da bb71 b9dc 629b cacc
00000040 dffb 192f 12b3 14d8 b7f4 9e36 528c 949f
00000050 5702 a722 622e ee44 b50a f75c d7d3 b2bd
00000060 7fcd 77b3 561a ea48 992a a77f f89a 2011
00000070 98b9 7004 0d65 2708 62e0 dac7 25bc 76da
00000080
steve@rei:~/code/md5coll> hexdump good2 | \
grep -3 "ceba b3eb e51d 153f fbf0 c6ef 08f5 abe4"
0000690 82fe 0804 830e 0804 0000 0000 0000 0000
00006a0 0000 0000 0000 0000 95ac 0804 0000 0000
00006b0 0000 0000 0000 0000 0000 0000 0000 0000
00006c0 ceba b3eb e51d 153f fbf0 c6ef 08f5 abe4
00006d0 adfd 0fac 6d23 800f 0ed1 b652 071f 4643
00006e0 2cb1 0533 7df8 df74 f147 76b6 035c 52fa
00006f0 4917 221c 5620 66da bb71 b9dc 629b cacc
--
0000710 5702 a722 622e ee44 b50a f75c d7d3 b2bd
0000720 7fcd 77b3 561a ea48 992a a77f f89a 2011
0000730 98b9 7004 0d65 2708 62e0 dac7 25bc 76da
0000740 ceba b3eb e51d 153f fbf0 c6ef 08f5 abe4
0000750 adfd 0fac 6d23 800f 0ed1 b652 071f 4643
0000760 2cb1 0533 7df8 df74 f147 76b6 035c 52fa
0000770 4917 221c 5620 66da bb71 b9dc 629b cacc
steve@rei:~/code/md5coll> hexdump bad2 | \
grep -3 "ceba b3eb e51d 153f fbf0 c6ef 08f5 abe4"
0000690 82fe 0804 830e 0804 0000 0000 0000 0000
00006a0 0000 0000 0000 0000 95ac 0804 0000 0000
00006b0 0000 0000 0000 0000 0000 0000 0000 0000
00006c0 ceba b3eb e51d 153f fbf0 c6ef 08f5 abe4
00006d0 adfd 0fac 6d23 800f 0ed1 b652 071f 4643
00006e0 2cb1 0533 7df8 df74 f147 76b6 035c 52fa
00006f0 4917 221c 5620 66da bb71 b9dc 629b cacc
--
0000710 5702 a722 622e ee44 b50a f75c d7d3 b2bd
0000720 7fcd 77b3 561a ea48 992a a77f f89a 2011
0000730 98b9 7004 0d65 2708 62e0 dac7 25bc 76da
0000740 ceba b3eb e51d 153f fbf0 c6ef 08f5 abe4
0000750 adfd 8fac 6d23 800f 0ed1 b652 071f 4643
0000760 2cb1 0533 7df8 df74 f147 76b6 835c 52fa
0000770 4917 221c 5620 66da bb71 39dc 629b cacc
steve@rei:~/code/md5coll>
steve@rei:~/code/md5coll> ./rsamd5 -len=1728 good2
reading upto 1728 bytes
Pre-mature state is: 137544438 2579640441 2786644843 118460404
steve@rei:~/code/md5coll> ./rsamd5 -len=1728 bad2
reading upto 1728 bytes
Pre-mature state is: 137544438 2579640441 2786644843 118460404
steve@rei:~/code/md5coll>
```

---

---

**Algorithm 4** Using md5coll to find colliding binary strings

---

```
freespace@Tabitha$ ./md5coll \  
137544438 2579640441 2786644843 118460404  
block #1 done  
block #2 done  
unsigned int m0[32] = {  
0x19de4a87, 0xd5c66a1c, 0x31acecfd, 0x6c34b288,  
0x91ab0705, 0x8275bffd, 0xf95d15c4, 0xd1864ac1,  
0x0534ed51, 0xc1739fff, 0x876eace3, 0xe3a92936,  
0xd57e3ef9, 0xa5a89527, 0x927c4fcb, 0xf4f04b76,  
0x3c8dacb2, 0xeed12126, 0x63633a54, 0x0e3cd525,  
0x26d0f096, 0x2d3dec1a, 0xf6eb5528, 0x9ef5584c,  
0x583adf4d, 0xa9ed9e0d, 0x2c19cf62, 0x9dd4b46e,  
0x4baa5c9b, 0xce0bdc4f, 0xeedaabd2, 0x1cc7fedf,  
};  
unsigned int m1[32] = {  
0x19de4a87, 0xd5c66a1c, 0x31acecfd, 0x6c34b288,  
0x11ab0705, 0x8275bffd, 0xf95d15c4, 0xd1864ac1,  
0x0534ed51, 0xc1739fff, 0x876eace3, 0xe3a9a936,  
0xd57e3ef9, 0xa5a89527, 0x127c4fcb, 0xf4f04b76,  
0x3c8dacb2, 0xeed12126, 0x63633a54, 0x0e3cd525,  
0xa6d0f096, 0x2d3dec1a, 0xf6eb5528, 0x9ef5584c,  
0x583adf4d, 0xa9ed9e0d, 0x2c19cf62, 0x9dd4346e,  
0x4baa5c9b, 0xce0bdc4f, 0x6edaabd2, 0x1cc7fedf,  
};  
freespace@Tabitha$
```

---

---

**Algorithm 5** good2.c after array substitution

---

```
#include <stdio.h>
unsigned int m0[32] = {
    0x19de4a87, 0xd5c66a1c, 0x31acecfd, 0x6c34b288,
    0x11ab0705, 0x8275bffd, 0xf95d15c4, 0xd1864ac1,
    0x0534ed51, 0xc1739fff, 0x876eace3, 0xe3a9a936,
    0xd57e3ef9, 0xa5a89527, 0x127c4fcb, 0xf4f04b76,
    0x3c8dacb2, 0xeed12126, 0x63633a54, 0x0e3cd525,
    0xa6d0f096, 0x2d3dec1a, 0xf6eb5528, 0x9ef5584c,
    0x583adf4d, 0xa9ed9e0d, 0x2c19cf62, 0x9dd4346e,
    0x4baa5c9b, 0xce0bdc4f, 0x6edaabd2, 0x1cc7fedf,
};
unsigned int m1[32] = {
    0x19de4a87, 0xd5c66a1c, 0x31acecfd, 0x6c34b288,
    0x11ab0705, 0x8275bffd, 0xf95d15c4, 0xd1864ac1,
    0x0534ed51, 0xc1739fff, 0x876eace3, 0xe3a9a936,
    0xd57e3ef9, 0xa5a89527, 0x127c4fcb, 0xf4f04b76,
    0x3c8dacb2, 0xeed12126, 0x63633a54, 0x0e3cd525,
    0xa6d0f096, 0x2d3dec1a, 0xf6eb5528, 0x9ef5584c,
    0x583adf4d, 0xa9ed9e0d, 0x2c19cf62, 0x9dd4346e,
    0x4baa5c9b, 0xce0bdc4f, 0x6edaabd2, 0x1cc7fedf,
};
int main(void)
{
    int i;
    for ( i = 0; i < 32; ++i)
    {
        if ( m0[i] != m1[i])
        {
            puts("Die world!");
            return 0;
        }
    }
    puts("Hello world!");
    return 0;
}
```

---

---

**Algorithm 6** bad2.c after array substitution

---

```
#include <stdio.h>
unsigned int m0[32] = {
    0x19de4a87, 0xd5c66a1c, 0x31acecfd, 0x6c34b288,
    0x91ab0705, 0x8275bffd, 0xf95d15c4, 0xd1864ac1,
    0x0534ed51, 0xc1739fff, 0x876eace3, 0xe3a92936,
    0xd57e3ef9, 0xa5a89527, 0x927c4fcb, 0xf4f04b76,
    0x3c8dacb2, 0xeed12126, 0x63633a54, 0x0e3cd525,
    0x26d0f096, 0x2d3dec1a, 0xf6eb5528, 0x9ef5584c,
    0x583adf4d, 0xa9ed9e0d, 0x2c19cf62, 0x9dd4b46e,
    0x4baa5c9b, 0xce0bdc4f, 0xeedaabd2, 0x1cc7fedf,
};
unsigned int m1[32] = {
    0x19de4a87, 0xd5c66a1c, 0x31acecfd, 0x6c34b288,
    0x11ab0705, 0x8275bffd, 0xf95d15c4, 0xd1864ac1,
    0x0534ed51, 0xc1739fff, 0x876eace3, 0xe3a9a936,
    0xd57e3ef9, 0xa5a89527, 0x127c4fcb, 0xf4f04b76,
    0x3c8dacb2, 0xeed12126, 0x63633a54, 0x0e3cd525,
    0xa6d0f096, 0x2d3dec1a, 0xf6eb5528, 0x9ef5584c,
    0x583adf4d, 0xa9ed9e0d, 0x2c19cf62, 0x9dd4346e,
    0x4baa5c9b, 0xce0bdc4f, 0x6edaabd2, 0x1cc7fedf,
};
int main(void)
{
    int i;
    for ( i = 0; i < 32; ++i)
    {
        if ( m0[i] != m1[i])
        {
            puts("Die world!");
            return 0;
        }
    }
    puts("Hello world!");
    return 0;
}
```

---

---

**Algorithm 7 Patch to original RSA MD5 Message Digest implementation**

---

```
--- rsamd5-orig.c      2005-11-21 15:02:35.000000000 +1100
+++ rsamd5.c          2005-11-26 15:32:18.000000000 +1100
@@ -362,6 +362,7 @@
#include <sys/types.h>
#include <time.h>
#include <string.h>
+#include <stdlib.h>
/* -- include the following file if the file md5.h is separate -- */
/* #include "md5.h" */
@@ -444,8 +445,9 @@
Prints out message digest, a space, the file name, and a carriage
return.
*/
-static void MDFile (filename)
+static void MDFile (filename, size)
char *filename;
+long size;
{
FILE *inFile = fopen (filename, "rb");
MD5_CTX mdContext;
@@ -458,11 +460,23 @@
}
MD5Init (&mdContext);
- while ((bytes = fread (data, 1, 1024, inFile)) != 0)
+ for(;;(bytes = fread (data, 1, 1, inFile)) != 0 && size;--size)
+ {
+ MD5Update (&mdContext, data, bytes);
+ }
+ if ( !size )
+ {
+ /* size was specified, just print out the states */
+ printf("Premature state is: %lu %lu %lu %lu\n",
+ mdContext.buf[0],mdContext.buf[1],
+ mdContext.buf[2], mdContext.buf[3]);
+ }
+ else
+ {
+ MD5Final (&mdContext);
+ MDPrint (&mdContext);
+ printf (" %s\n", filename);
+ }
+ fclose (inFile);
@@ -502,12 +516,12 @@
MDFile ("foo");
}
-void main (argc, argv)
+int main (argc, argv)
int argc;
char *argv[];
{
int i;
-
+ long size=-1;
/* For each command line argument in turn:
** filename -- prints message digest and name of file
** -sstring -- prints message digest and contents of string
@@ -525,7 +539,14 @@
MDTimeTrial ();
else if (strcmp (argv[i], "-x") == 0)
MDTestSuite ();
- else MDFile (argv[i]);
+ else if (strstr(argv[i], "-len="))
+ {
+ size = strtol(argv[i]+strlen("-len="),NULL,10);
+ printf("reading upto %lu bytes\n", size);
+ }
+ else MDFile (argv[i],size);
+
return 0;
}
/*
```