MD5 To Be Considered Harmful Someday

Dan Kaminsky
Senior Security Consultant, Avaya
dan@doxpara.com

Abstract

Joux and Wang's multicollision attack has yielded collisions for several one-way hash algorithms. Of these, MD5 is the most problematic due to its heavy deployment, but there exists a perception that the flaws identified have no applied implications. We show that the appendability of Merkle-Damgard allows us to add any payload to the proof-of-concept hashes released by Wang et al. We then demonstrate a tool, Stripwire, that uses this capability to create two files – one which executes an arbitrary sequence of commands, the other which hides those commands with the strength of AES – both with the same MD5 hash. We show how this affects file-oriented system auditors such as Tripwire, but point out that the failure is nowhere near as catastrophic as it appears at first glance. We examine how this failure affects HMAC and Digital Signatures within Digital Rights Management (DRM) systems, and how the full attack expands into an unusual pseudo-steganographic strikeback methodology against peer to peer networks.

I. INTRODUCTION

THE modern application of cryptographic principles is actually quite primitive – not in its complexity, but in the way the complexity has been managed. Independent primitives such as hashes and ciphers completely specify the behavior of a limited set of aggressively audited algorithms. Each trusted implementation is chosen to be entirely functionally equivalent to one another; choosing one over another is to have no impact on what the user (legitimate or otherwise) can do. Deviations between the chosen algorithms are limited to speed of operation, some mild key and block size constraints, and a vaguely understood ”security level” of the underlying mathematics. It is this last fear – that even after all our auditing, something will still get through – that drives adherence to the primitive specification. Through this we acquire modularity, the ability to swap out the broken part for something that works.

But just because we can do something doesn't mean we will. Joux and Wang have made it plainly clear that MD5 has serious problems. This shouldn't come as much surprise; Dobbertin's work [1] almost a decade ago made it clear that this was coming. Yet even now, there are those who have hinted that there isn't any applied risk, and that the vulnerabilities are purely theoretical. Outside of FIPS's unwillingness to certify MD5, there is no apparent push to migrate away from MD5 as we once did for its predecessor, MD4.

The attacks discovered are indeed obscure. But completely theoretical? No. Even given what little data has been released – code implementing the attack isn't even public yet – sufficient information has been released to piece together a rudimentary proof of concept tool that demonstrates, at minimum, that the selection of MD5 exposes new and potentially deeply undesirable functionality above and beyond what the one-way hash primitive specifies. The tool, Stripwire, implements some of the attacks described herein.

That being said, this paper is not a ”smoking gun” indictment of MD5. I've taken great pains to include the caveats of each vulnerability, as it is far too easy to overestimate the risks described in this paper. It is for that reason I am not saying ”today”, or ”any day now”. The title states ”someday” for a reason. There are dots going back ten years as to the risk of MD5. Here are a few more, in the hopes that they will start to be connected.

II. MD5 HOWTO

For a detailed description, look elsewhere [2] [3]. Put simply though, MD5 is an implementation of a one-way hash, by which an arbitrary amount of data may be reduced to a 128 bit fingerprint of what went in. The hash is one way when it's simple to compute the hash from arbitrary data, but difficult – in a ”computationally infeasible” sense – to reverse the process, finding data that matches a particular hash. The hashing process needs to be resistant to the point where two datasets cannot even be created for the express purpose of ”colliding” – having the same hash value. These cryptographically strong one way hashes are quite useful when we want to store summaries of data, and retain the ability to recognize that data at a later time, without actually having to keep a copy of the original data around or needing to worry about other people being able to pretend they have a copy of the original data.

III. The Discovery: Joux's Multicollision Attack

For MD5 (and actually, a number of popular hashing algorithms, SHA-1 not included), there exists a weak class of input data for which subtle changes can be silently introduced without causing apparent changes in the final MD5 hash. Capacity is not huge – of the two 128 byte proof-of-concept files released by Wang [4], only six bits differ. But many ”doppelganger” sets can be computed, each of which may be swapped out with the other at no effect to the resultant hash. The sets are two MD5 blocks long.

This is a fundamentally new mode of operation for MD5.

IV. Extending the Attack

To see how this relatively obscure new mode can cause problems, it is necessary to understand how MD5 works. In what's referred to as a Merkle-Damgard construction, MD5 starts with an arbitrary initial state, 128 bits in length. 512 bits of input data are ”stirred” into this 128 bit state, with a new, massively shuffled 128 bit value as the result. 512 more bits are constantly stirred in, over and over, until there's no further data. One more round of stirring is done to embed the precise size of the file, and the final 128 bit value is christened the MD5 hash.

Now, amongst the cryptological community, there is a well known failure mode to the this particular construction: If, at any point in the cascade, two different datasets are stirred into equal 128 bit values, arbitrary data can be appended to both datasets and their hashes will remain equal. In mathematical terms (with minor assumptions forgiven), if md5(x) == md5(y), then md5(x+q) == md5(y+q).

It's relatively straightforward to see why this works: Files are read in 512 bits at a time, with each block summarized into only what can fit inside the 128 bit value. Once two deviant datasets collide to the same 128 bit value, anything added on after the fact is too late – that difference will remain permanently cloaked.

The original attack gives us our two deviant datasets. This extension shows us how we can append arbitrary data after the datasets and still retain collision. Stripwire demonstrates how we can convert this collision into an applied attack.

V. Stripwire

We begin by defining two files, ”vec1” and ”vec2”, as the proof-of-concept test vectors released by Wang. Vec1 and Vec2 have the same MD5 hash, but differ by six bits out of 1024. We also define ”payload” as some arbitrary string of commands to be executed. The ”encrypted payload” is simply the AES encrypted representation of payload, using the SHA-1 of vec1 as the key. (It is useful to note that while vec1 and vec2 do share the same MD5 hash, they do not share the same SHA-1 hash.)

We now define two more files, ”Fire” and ”Ice”. Fire is simply vec1 with the encrypted payload appended to it, while Ice is vec2 with the encrypted payload attached. Only six bits separate Fire and Ice, but this small deviation is critical. Fire contains vec1, which can be easily hashed to acquire the key to the encrypted payload. Ice contains vec2, which SHA-1’s out to a useless value that fails to decrypt the payload. So while Fire easily exposes the means to burn the system, Ice’s payload remains frozen in its AES-enforced shell.

Radically deviant behavior – Same MD5 hash. Returning to the math, the encrypted payload is q; it is the constant payload appended to the x and y of colliding sets. Through this mechanism, Ice and Fire can be exchanged at will, and as far as MD5 is concerned, nothing ever happened.

This is not theoretical. It looks like this:

A. Demo

Stripwire itself has been designed to be as readable as possible; for some readers its source code will be much better documentation than this paper. For those seeking to reimplement the attack from this document alone, the two test vectors are as follows:

```
  $vec1 = h2b("
d1 31 dd 02 c5 e6 ee c4 69 3d 9a 06 98 af f9 5c
2f ca b5 87 12 46 7e ab 40 04 58 3e b8 fb 7f 89
55 ad 34 06 09 f4 b3 02 83 e4 88 83 25 71 41 5a
08 51 25 e8 f7 cd c9 9f d9 1d bd f2 80 37 3c 5b

d8 82 3e 31 56 34 8f 5b ae 6d ac d4 36 c9 19 c6
dd 53 e2 b4 87 da 03 fd 02 39 63 06 d2 48 cd a0
e9 9f 33 42 0f 57 7e e8 ce 54 b6 70 80 a8 0d 1e
c6 98 21 bc b6 a8 83 93 96 f9 65 2b 6f f7 2a 70
");

$vec2 = h2b("
d1 31 dd 02 c5 e6 ee c4 69 3d 9a 06 98 af f9 5c
2f ca b5 07 12 46 7e ab 40 04 58 3e b8 fb 7f 89
55 ad 34 06 09 f4 b3 02 83 e4 88 83 25 f1 41 5a
08 51 25 e8 f7 cd c9 9f d9 1d bd 72 80 37 3c 5b

d8 82 3e 31 56 34 8f 5b ae 6d ac d4 36 c9 19 c6
dd 53 e2 34 87 da 03 fd 02 39 63 06 d2 48 cd a0
e9 9f 33 42 0f 57 7e e8 ce 54 b6 70 80 28 0d 1e
c6 98 21 bc b6 a8 83 93 96 f9 65 ab 6f f7 2a 70
");
```

A line space has been inserted between the two MD5 blocks, and bytes with deviant bits have been highlighted. For example, the byte set to ”87” in vec1 is set to ”07” in vec2. It’s worth noticing that the changes within each vector are repeated, in the same position, between their first block and their second block.

Now, onto our payload. Our payload to be encrypted may be of arbitrary size; for the purposes of this paper, we will demonstrate a bare-bones application that opens a pseudoshell on an arbitrary port.

```
$ cat backlash.pl
#!/usr/bin/perl
# Backlash:  Open a pseudoshell on port 50023
#   Author:  Samy Kamkar, www.lucidx.com

use IO;
while(1){
   while($c=new IO::Socket::INET(LocalPort,
         50023,Reuse,1,Listen)->accept){
            $˜->fdopen($c,w);
            STDIN->fdopen($c,r);
            system$_ while<>;
   }
}
```

First, we generate Fire and Ice.

```
$ ./stripwire.pl -v -b backlash.pl
fire.bin: md5  = 4df01ec3a18df7d7d6cdf8e16e98cd99
ice.bin:  md5  = 4df01ec3a18df7d7d6cdf8e16e98cd99
fire.bin: sha1 = a7f6ebb805ac595e4553f84cb9ec40865cc11e08
ice.bin:  sha1 = 85f602de91440cd877c7393f2a58b5f0d72cbc35
```

Note, their md5sum’s match, but not their sha1sums. And, of course, they share the same filesize.

```
$ ls -l fire.bin ice.bin
-rw-r--r--    1 kaminsky mkgroup_      496 Nov 30 20:50 fire.bin
-rw-r--r--    1 kaminsky mkgroup_      496 Nov 30 20:50 ice.bin
```

Binary comparison cannot be fooled, however.

```
$ diff fire.bin ice.bin
Files fire.bin and ice.bin differ
```

Stripwire contains the execution harness for Fire and Ice. When we run it against Ice...

```
$ ./stripwire.pl -v -r ice.bin
Unable to decrypt file: ice.bin
```

Failure. Fire is another story:

```
$ ./stripwire.pl -v -r fire.bin &
[1] 1420

$ telnet 127.0.0.1 50023
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is ’ˆ]’.
cat /etc/ssh_host_dsa_key_demo
-----BEGIN DSA PRIVATE KEY-----
MIH5AgEAAkEAlcTshGgpYY0eQgRBJRyQCrBDgXhFWFTbxazsgbrKiebh1aal4ET6
vPYZ7/OlPbrKxwMnX5mcEHywmEhOcK00pwIVAJyQ0ZlkpRPr2eJWz/ECgr1XgUvP
AkBWeUy6MJHApO5sF+T0V7vs319fGvw0j8dthueQ2pAZHJl063SC2n9JkaMZRHEn
J7c04xMEHnFdmIvxTNFCavKZAkEAieVtNTFNNV7SIf0m4z60mJ1Hz3zj50R7ih1S
SxPon+IxzKsoAEP9JkyjS67+HBQGpowxNuukOFaqDwl1gclGfwIVAJuPpSn6yj2e
z5m7aTzZ72B131h8
-----END DSA PRIVATE KEY-----
```

VI. Caveats

It should come as no surprise that the primary applied target for the Stripwire tool is the highly popular ”Tripwire” file system auditing tool. Although Tripwire can configured to use more trustworthy algorithms, under common configurations it works by collecting MD5 hashes of every system file contained within the file system and alarming if any of those hashes change. The base security presumption is that as long as the file system doesn't change, neither will the behavior of the system running on top of it.

Stripwire makes it trivial for an attacker to swap out the harmless Ice for the arbitrarily dangerous Fire, with Tripwire none the wiser. So does this mean Tripwire is fundamentally broken? The short answer, no, absolutely not. The longer answer is where things get interesting.

We begin by looking closer at Tripwire's base presumption – yes, security engineers use Tripwire to detect unauthorized changes in the file system, but altering the file system is not the only way operations can be affected. The file system doesn't fully define an operating environment any more than laws fully define a legal system. Any number of external sources can alter behavior. Faults amidst its files are but one path, and not necessarily the best one. An entire branch of exploit research focuses on memory-only attacks that use the network as their injection vector and alter only the in-RAM kernel or library structures to support remote control of the OS. The disk is never touched; all evidence bleeds away the moment the plug is pulled by a naive forensic analyst.

And of course, systems do not need to be networked to exhibit deviant behavior with a constant software load. Anything from CPU speed to motherboard temperature sensors to the particular date emitted by the RTC (Real Time Clock) can be used to select between completely different sets of instructions. Systems can be even be configured to alter their behavior randomly. What matters is what the system is programmed to do, and that's the second problem: Tripwire doesn't tell you that you can trust something; only auditors can do that. It only says if you could trust it before, you still can now. For Stripwire to pose an actual threat to a deployed environment, not only would Ice need to be added to the trusted list of MD5-monitored files, but so too would the Stripwire execution harness itself. That is an unlikely circumstance.

So most uses of MD5, even by Tripwire, remain secure – under the present threat regime, at least. There still remains a critical blind spot in anything that uses MD5; to pick one example, this is a fantastic channel for a group of malicious developers to submit innocuous and undecryptable content to their auditors for approval, and then once it's acquired to swap in a self-decrypting and unaudited payload. Audits against the shipped code would show the same MD5 hashes, and all would appear well.

Ultimately, MD5 cannot be depended upon to protect against a bait and switch, and neither can anything that depends on it.

VII. Digital Signatures and DRM

Digital Rights Management, or DRM, has become a catch-all term for a extensive reimagining of issues not simply technical, but legal, political, and economic as well. The latter three have effectively driven the concept of a mutually trusted ”third party attester” into technology that has traditionally operated on a ”dumb automaton” model of command/execution. Third party attestation allows a third party to control the precise manner in which a system should operate, independent of mere technological capacity. Cryptographic primitives are chained together in DRM systems to link grantable resources to the externally provided objects that provide the granting.

DRM systems with MD5 as part of their chain could conceivably face problems, even with they never hash data directly. All three major digital signature algorithms – RSA, DSA/ElGamel, and Elliptical Curve – are almost universally used in a mode where they do not sign data directly, but rather sign a hashed representation of the data. (Asymmetric algorithms are quite slow; this manuever makes it realistic to sign arbitrarily large files.) Often, the hash algorithm of choice is MD5. Identical input yields identical output – if two files have the same hash, they'll both verify against the same signature. So, a key constraint of the digital signature primitive, that no other data could survive signature verification save for the data that was originally signed, cannot be met.

The appears to be only limited vulnerability to this in open deployment. Microsoft's Authenticode technology, used within its browser to limit executable content within web page to signed documents, does indeed use (or at least allow) MD5 hashes to be signed. It would be trivial to sign something innocuous and then actually release something malicious. But the security model of Authenticode has always been one of legal accountability – having someone to sue – and not of technical restriction. Indeed, the amount of abjectly destructive ”spyware” tunneled to user machines through Internet Explorer is astonishing.

It is worth noting that probably the widest-deployed hardware that employs digital signatures for third-party attestation, the Microsoft X-Box, uses SHA-1 as its hashing algorithm and not MD5 [5]. So it is not vulnerable. But it's also worth noting that, had Microsoft selected MD5 instead of SHA-1, their use of a 2048 bit key for their RSA signature would have been completely irrelevant.

VIII. Multicollisions Unleashed

Interestingly enough, none of what's been discussed already actually requires the full attack discovered by Joux and Wang. Thus far, everything has been based only on the ability to append arbitrary data to Wang's test vectors. But failures inside cryptographic primitives, even very small ones, tend to lead to slowly discovered catastrophic failures. MD5 does not seem to be an exception to this rule.

We can do much more with the actual multicollision attack. The test vectors collide only when stirred into the default initial state for the MD5 algorithm; the attack itself works against any arbitrary state. The upshot here is that we can not only append arbitrary data but prepend it as well. Presently, Fire and Ice required a dedicated external execution harness, which could arouse suspicion. With prepending available, a correctly formatted binary executable could be synthesized that would self-analyze and branch appropriately depending on which vector was contained within.

In addition, being limited to the MD5 initial state means only hashes calculated on a per-file basis can be made to collide; a full disc or partition sum will come across the doppelganger set at a vastly different initial state and fail to collide. With the full attack, we could specify our colliding blocks against the MD5 state that would be found during a full disk or partition hashing operation. Of course, then the colliding set we generated wouldn't collide on a per-file basis. Thus far, we can only adapt to a single MD5 state at a time.

IX. HMAC

Most observers have written that the HMAC, or Hashed Message Authentication Code, construction is entirely immune to the multicollision attack. They are mostly correct, just not entirely correct. HMAC is a method of taking an arbitrary hashing algorithm, like MD5, and introducing a secret to it such that only someone with that secret can synthesize or verify the correct hash for some arbitrary input. In simple terms, HMAC is a mechanism for altering the initial state according to some password. In more precise terms, HMAC does the following:

```
Inner = MD5(Key XOR 0x36 + Data)
Outer = MD5(Key XOR 0x5c + Inner)
HMAC-MD5 = Outer
```

There are three things going on here:

1. A key is prepended to the data being hashed.
2. Additional noise is added to the user provided key.

3. Two rounds of hashing are used instead of one, with the noise varying between the two rounds.

There's a fair amount of defensive cryptosystem design inside of HMAC – it's an avowed goal of the algorithm to still function even if small faults are found in the underlying hash. But for all its defensive operation, the Data portion is only invoked once, prepended with a key-derived block. This new block creates a new 128 bit state for Data to be stirred into, and this state diverges substantially from our generic MD5 initial state.

But the multicollision attack works against any state, not just the generic one. That means it's straightforward, given the key, to adapt to the new system state and cause a collision in the inner hash. Once inner has collided, outer must as well, as its getting the same input from either dataset. (If the outer hash also concatenated the Data portion, the attack would entirely fail, because it's thus far impossible to adapt to two separate MD5 states.)

This is the first known method of creating two datasets that collide under HMAC-MD5. Once again, though, the caveats are deep.

From one perspective, saying HMAC is insecure when the key is known is a little like saying AES is insecure when the key is known – the whole point is that the key is unknown to the attacker. It's quite arguable that the MAC primitive, like anything else with a key, is allowed to collapse if the key leaks. The basic idea of open crypto design, after all, is to migrate all secrecy and security out of the algorithm and into the key. Is it really fair to complain when, given this design, risks show up with the key being lost?

Perhaps. For all the analysis in this paper, the multicollision attack against MD5 remains relatively weak, with special circumstances required for an attack to succeed. For Tripwire to be compromised, the initial trust database needed to be infected. For digital signatures to be affected, something only apparently innocuous needed to be signed first. In both cases, the use of MD5 opened up a small threat vector; would HMAC have changed this? If an attacker has access to a system such that files may be altered, he probably also has access to whatever HMAC key Tripwire had been reconfigured to use (assuming the entire contents of the file system aren't being streamed over the network to an uncompromised host). So HMAC doesn't change the threat scenario for Tripwire. And for the digital signature bait and switch, the attacker has to have the HMAC key to create a signable payload for the third party attester.

Ultimately, as most uses of MD5 are immune to the multicollision vulnerability, so too are most uses of HMAC-MD5. But when MD5 does experience an applied threat, HMAC-MD5 provides limited if any protection.

X. Strikeback: Traitor Tracing

Security is a battle between attackers and defenders, and defenders do not necessarily need to cede the ground of cryptographic exploitation to attackers alone. A research path, known as ”Strikeback”, examines the mechanisms by which a defender under attack can exploit weaknesses in his attackers to defend his systems.

There are strikeback implications to this MD5 research.

The proof of concept for Stripwire was simple: Take an audio file encoded in the MPEG-1 Layer 3 (MP3) format. Append it to both vec1 and vec2. Note that the agglomerated files both play flawlessly and identically. This wasn't a surprising result; MP3 is a bitstream format and as such is highly resistant to so-called ”junk data” infecting the datastream. But this was the first proof that two files with bit-differences and the same MD5 hash could still function correctly given a cooperative execution harness, and led to the basic design of Stripwire.

It also yielded an MP3 file that contained an extra bit of information – whether vec1 or vec2 had been prepended. A single bit is not useful. But we are not constrained to a single bit.

Wang has disclosed that, given an arbitrary MD5 system state, her implementation is capable of finding a multicollision-capable set after approximately one hour of computation, with one doppelganger computable every fifteen minutes after that. It is well within the realm of feasiblity to compute 16 sequential multicollision sets, each adapting to the MD5 state emitted by the previous, with 256 (or 28̂)computed doppelgangers for each set. Now, instead of the single bit of information represented by the choice between vec1 and vec2, we have 8 bits of information per prepended block – and there are 16 blocks. This yields space for a 128 bit signature, and things just got much more interesting.

A. MP3

Consider the problem of tracing the path of an MP3 file as it winds its way through a peer to peer network. Since MP3 files are error-resilient, one could connect a custom client to the network that prepended a unique 128-bit serial number to every song transmitted. Every second-level copy would now be individually tagged, and it'd be possible to trace every file on the network back to the second-level host that retrieved it from the custom file server. Adding a deviating serial number to each file transmitted would normally cause problems, as both the search algorithms and file integrity checks on P2P networks tend to be MD5 centric. But since the serial number is represented in a form that MD5 is blind to, nothing fails – except perhaps some of the opacity of the P2P network.

That's not to say there aren't countermeasures. The serial number is easy to detect, can be trivially stripped, is simple to alter, and can be rendered inoperable simply by switching the network to another hashing algorithm. But even here, there are caveats – detection may be simple, but eliminating the serial number entirely will yield a different hash value, subtlely breaking the network's ability to coalesce all identical payloads. And while it's possible for hosts on the P2P network to ”mix and match” doppelganger sets from several hosts, it's relatively straightforward to identify a cryptographically secure subset of the 21̂28 possible serial numbers that makes it impossible for users to synthesize valid serial numbers through any other means of acquiring them from a first or second generation source. And finally, peer to peer networks are still networks and as such are vulnerable to the greatest caveat of network effects: Even after faults are identified, so many nodes may depend on the faulty behavior that the value of the network is decreased more by fixing the fault than it is by suffering its continued presence.

There is one special case on P2P networks – some designs allow a file to be acquired in pieces from several different nodes. One solution to this is to seed a file with serial numbers across its entire body, perhaps three sets every 128 kilobytes. This is a much more compute-intensive operation, though, since the multicollision sets must be computed on a per-file basis as different data will preceed each group of doppelgangers.

B. Executables

Barring some of the more creative and noticably illegal designs which infect MP3 files with executable content, it's not possible for MP3-embedded signatures to yield any more evidence except for what they present by their existence on any number of hosts.

Actual executables are another story. They are generally quite full of undocumented and undocumentable functionality, much of it inserted by a compiler. (Thus the limits of auditors – they may be able to read source, but how many can read what the compiler actually emits? Because that's what the system ultimately needs to trust.) Particularly if an executable is aggressively protected from public distribution, there can be no expectation of publically safe behavior (in fact, that's generally why aggressive protections are instituted in the first place. That, and profit motives.) It would be quite irresponsible to embed code that erased hard drives or flooded networks...

But why not locate the source of the leak?

128 bits is a fair amount of capacity – English text only takes 1.3 bits per character, compressed – and it'd

be reasonable to quadruple that if needed. Before distributing an illicitly acquired executable, an attacker is likely to test it during their packaging process. During the testing process, the executable installer could be configured to collect PII (Personally Identifiable Information) from across the file system. The 128 to 512 most valuable bits of information would be locally transformed into the requisite MD5-blind series of doppelgangers, and injected back into the installer upon its exit, before mass distribution could take place. The range of acquirable data is extensive – potential sources include:

1. Network data – IP address, DNS name, default name server, MAC address
2. Browser Cookies, Caches, and Password Stores – Online Banking, Hotmail, Amazon 1-Click
3. Cached Instant Messenger Credentials – Yahoo, AOL IM, MSN, Trillian
4. P2P Memberships – KaZaA, Gnutella2
5. Corporate Identifiers – VPN Client Data / Logs
6. Shipped Material – CPU ID, Vendor ID, Windows Activation Key
7. System Configurations – Time Zone, Telephone API area code
8. Wireless Data – MAC addresses of local access points
9. Existence Tests – Special files in download directory

Also possible, but legally problematic, would be acquiring not just one hop's worth of data but watching the executable as it travels across large networks, containing identifying information for as many previous hops as possible. Capacity becomes a problem, as it does with IP's ”Record Route” option, but we can handle it by dynamically reducing resolution (the RRDTool approach) or by simply keeping an overflow counter (what IP does).

And as before, the trafficked file would maintain the same MD5 hash no matter the data hidden inside. In a very unique sense, this is a steganographic strategy aimed not at the human analyst but at the precise internals of the MD5 algorithm. It's quite effective.

XI. Conclusions

The point is not that MD5 has collapsed. It hasn't. The point is that there's a very clear trend regarding the security level of MD5, and it isn't good. It is now undeniable is that the selection of MD5 matters – the constraint that deployed implementations of the one-way hash primitive be functionally identical has been broken. The failures detected are not merely algorithmic or theoretical, rather new capabilities above and beyond what the primitive specifies are made available by the selection of MD5. It is not expected that this paper will cause a precipitous decline in the use of MD5; that will probably occur when a means of silently introducing single-bit errors in arbitrary (rather than chosen) MD5 payloads is discovered.

But in the security community, we tend to complain about the ”phase change” nature of our systems, that suddenly collapse from secure to insecure on the discovery of a ”zero day” exploit. The phase change for MD5 isn't here yet, but it will come, someday. Nobody should be surprised when that day arrives.

References

[1] Hans Dobbertin, “Cryptanalysis of md5 compress,” 1996, `http://citeseer.ist.psu.edu/68442.html`.

[2] Philip Hawkes, Michael Paddon, and Gregory G. Rose, “Musings on the wang et al. md5 collision,” Cryptology ePrint Archive, Report 2004/264, 2004, `http://eprint.iacr.org/`.

[3] Stefan Lucks, “Design principles for iterated hash functions,” Cryptology ePrint Archive, Report 2004/253, 2004, `http://eprint.iacr.org/`.

[4] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu, “Collisions for hash functions md4, md5, haval-128 and ripemd,” Cryptology ePrint Archive, Report 2004/199, 2004, `http://eprint.iacr.org/`.

[5] Andy Green Peter Barth, Jeff Mears, "Project b (hacking) overview," 2004, http://www.xbox-linux.org/docs/projectboverview.html.