MD5 To Be Considered Harmful Someday

Dan Kaminsky

Senior Security Consultant, Avaya
dan@doxpara.com
06-Dec-2004

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. If everything implements the same specification, we can swap out a broken implementation for a correct one.

But just because we can do something doesn't mean we will. Joux [1] and Wang [2] have made it plainly clear that MD5 has serious problems. This shouldn't come as much surprise; Dobbertin's work [3] 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 [4] [5]. 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 and Wang's Multicollision Attack

For MD5 (and actually a number of popular hashing algorithms, SHA-1 not among them), it is possible to compute particular classes 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, 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. Because it's possible to compute new blocks on demand, a generic "antivirus" style colliding block detector isn't possible. It may be possible to generate a custom weak class detector. The ability to generate colliding datasets exposes 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. 64 bits more are appended to the data stream to explicitly reflect the amount of data being hashed with another round of MD5 being done if need be (if there wasn't enough room in a previous round to hash in that 64 bits), and the final 128 bit value after all the stirring is complete 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, using the "+" sign to refer to concatenation and assuming length(x) and length(y) both evenly divide into the 64 byte blocksize of MD5, if $md5(x) = md5(y)$, then $md5(x + q) = md5(y + q)$.

It's relatively straightforward to see why this occurs: 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 – MD5 may be a chaotic and nonlinear function, but from the same seed, the chaotic linearity between the two datasets will remain forever synchronized with the early difference forever 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 in this case 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 can be run through the SHA-1 hash but yields 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.[1]

Fire burns. Ice remains frozen. Fire and Ice have the same MD5 hash. Returning to the math, the encrypted payload is q; it is a 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: