Hash Pile Ups: Using Collisions to Identify Unknown Hash Functions

R. Joshua Tobin and David Malone

11 October 2012

Hash Functions

We are talking about hash functions for consistent assignment. For example,

- Hash tables,
- Network balancing packets (CEF, LAG, ECMP),
- Service load balancing (BIG-IP),
- Packets to CPUs (Microsoft RSS),
- etc.

These are not usually cryptographic strength!
Collisions relatively easy to find.

Outline

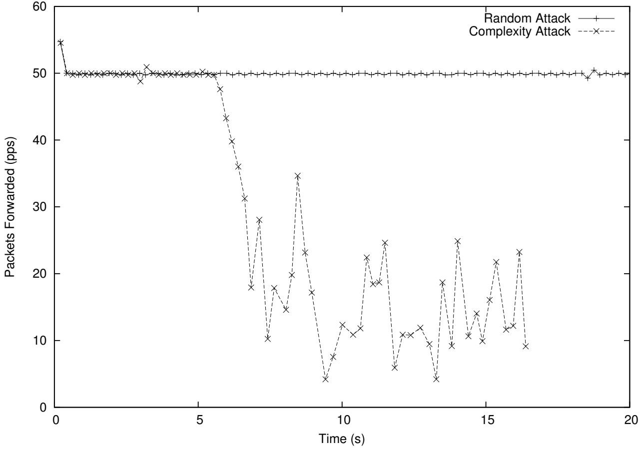
1. Background motivation.
2. Idea — learning and generating collisions.
3. 3 examples
   - 3.1 the hash,
   - 3.2 the attack,
   - 3.3 the results.
4. Conclusion.

There is an analysis of each attack in the paper.

Background Motivation

- Algorithmic Complexity Attacks (Crosby and Wallach, 2003).
- Some algorithms have different typical and worst case.
- Attack by choosing input to be worst case.
- Can be applied to hash tables, sorting, string matching, ...
- Hashes are canonical examples.

Demonstration attack

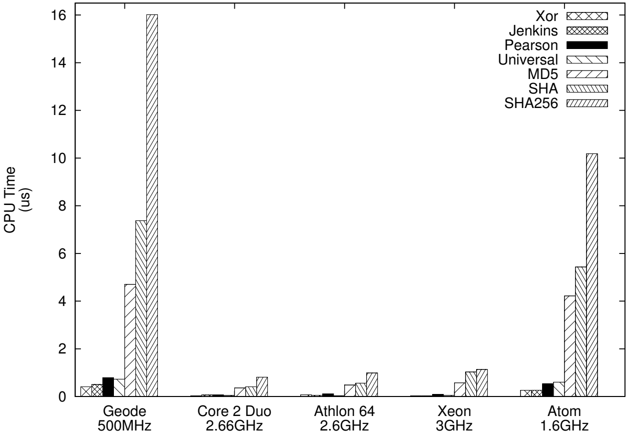


How to Fix?

- In general use algorithm with good worst case.
- Hash functions too useful though.
- Using crypto-strength hashes often too slow?
- What happens if the hash used is a secret?

Choose your hash randomly from a family on startup.
(Advisories still being released on this issues.)

Hash Costs



CPU Time
(us)
0
2
4
6
8
10
12
14
16
Xor
Jenkins
Pearson
Universal
MD5
SHA
SHA256
Geode
500MHz
Core 2 Duo
2.66GHz
Athlon 64
2.6GHz
Xeon
3GHz
Atom
1.6GHz

Idea — Learning from collisions

1. You usually can't observe hash output.
2. You can often observe collisions (e.g. time hash lookups, processing time, reordering, traceroute, server IDs, . . .).
3. By design, your hashes should have different collisions.
4. Observing collisions leaks information about hash in use

Can we use this to identify the hash function or generate collisions?

Example 1: Small Hash Family

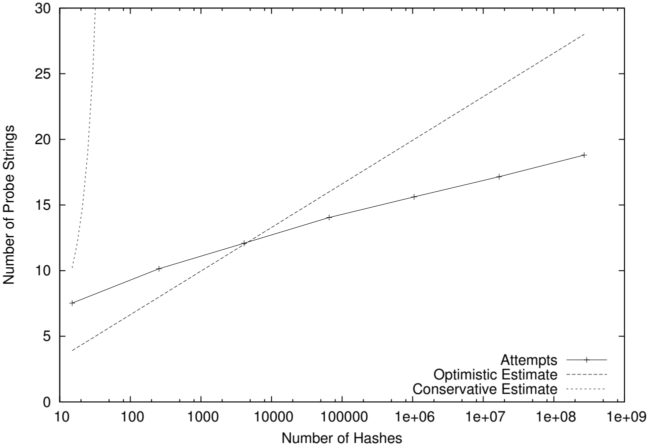
1. Often the hash is keyed by an integer or a few bits.
2. Suppose the number of hashes is small enough to iterate through.
3. For example, Bob Jenkins's hash in RFC 5475.
4. Use 4 bits of output (e.g. 16 routes).

Example 1: Small Hash Family

Attack:

1. Make a list of all hashes.
2. Find two colliding inputs (Birthday Paradox).
3. Remove hashes that do not collide on these inputs.
4. Repeat until one hash left.

Example 1: Small Hash Family



Example 2: Pearson's Hash

In 1990 Pearson proposed a neat, fast, randomly keyed hash, using a random permutation T of a byte and xor ($\otimes$).
To hash a string of bytes:

1. $h \leftarrow 0$
2. foreach ($byte[i]$)
   $$h \leftarrow T[byte[i] \oplus h]$$
3. return h

Family is really big — 256!

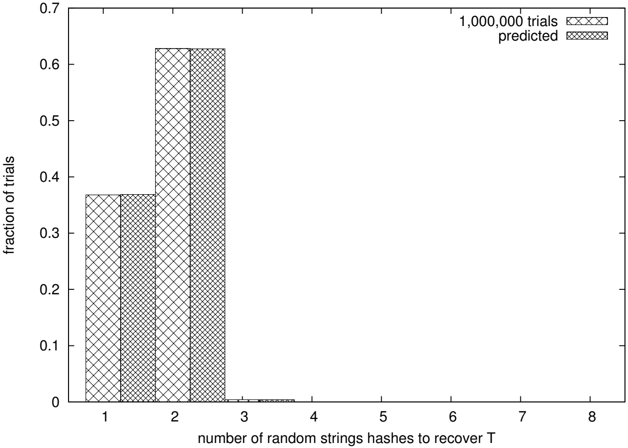
Example 2: Pearson's Hash

Attack: Recover the permutation.

1. Insert all strings *x000...0* and *0y00...0*
2. Algebra: collide in pairs (a, b) where $T(a) = T(0) \otimes b$.
3. From collisions, we know pairs (using 2*256 strings).
4. $T(0)$ is remaining unknown (small family, get in 256+small strings).

Attack generalises to replacing bytes and xor with any group.

Example 2: Pearson's Hash



Example 3: Toeplitz Hash

Microsoft have a standard for network cards to hand off packet to CPUs (RSS). The key K is a longish bit string.

1. $r \leftarrow 0$
2. foreach bit b in input
   if (b $==$ 1) $r \leftarrow r \otimes$ left-most 32 bits of K
   shift K left 1 bit position
3. return r

In practice you use 1–7 bits and might pass through a lookup table to choose CPU.

Example 3: Toeplitz Hash

Attack: It's linear over $\mathbb{Z}_2$, use some linear algebra.

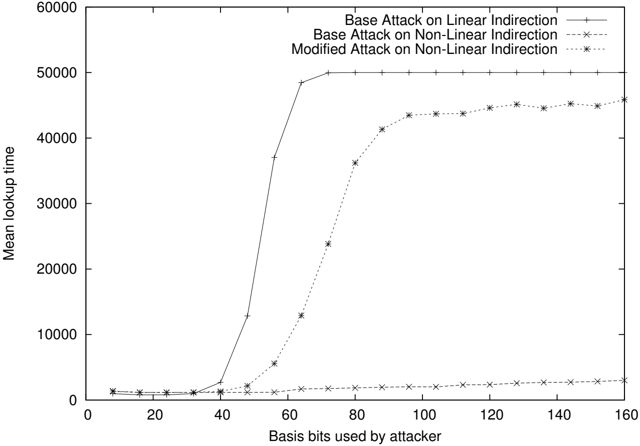
1. Choose the bits of the input you control. Set one to zero at a time.
2. Group the bits according to which collide $(E_1, \ldots, E_l)$.
3. For any even-sized subsets $E'_1, \ldots, E'_l$ of $E_1, \ldots, E_l$

$$h\left(x + \sum_{e \in \bigcup E'_i} e\right) = h(x) + \sum_{e \in \bigcup E'_i} h(e) = h(x),$$

4. So every even-sized subset collection gives a collision.

Can work with other linear functions too, but more effective for low index.

Example 3: Toeplitz Hash



Conclusion

1. Algorithmic Complexity Attacks.
2. For hashes, choosing from a family is useful.
3. However, collisions leak information.
4. Means you need to choose family carefully.
5. Small family is bad.
6. Structure like linear or group is bad.