Denial of Service via Algorithmic Complexity Attacks

Scott Crosby and Dan Wallach
Presented by Eddie Aftandilian

Problem

- Frequently used data structures and algorithms have good average-case running time but poor worst-case running time
  - Hash tables
  - Binary trees
  - Quicksort
- If we construct malicious input, we can DoS a system with a small amount of input

Problem

- Focus on hash tables in this paper as a specific example of this type of attack
Hash tables

- 2 ways to get a collision:
  - Objects hash to same value – hash collision
  - Objects hash into same bucket – bucket collision
- Attacker must compute objects that will eventually collide, either by a hash collision or a bucket collision
- Hash table must also accept enough input for worst-case behavior to manifest

Constructing an attack

- Must analyze source code of program to be attacked
  - What hash table implementation?
  - What hash function?
  - Bucket count?
  - Does external, untrusted input get fed into this hash table?
Hash collisions vs. bucket collisions

- Bucket count smaller than size of hash-value space, so easier to find collisions
  - But typically bucket count is not static
  - Can compute collisions for each possible bucket count, but quickly becomes more difficult than hash collisions
- So focus on hash collisions

Hash collisions

- But wait! Aren't hash collisions difficult to find?
  - Only for cryptographic hash functions (SHA-1, MD5)
  - Most hash functions used for hash tables are not cryptographically secure, often very simple
    - Ex. XOR input together in 32-bit chunks
    - These are easy to find collisions for

Squid

- Caching web proxy, widely used
- Uses hash table to determine whether a requested object is cached
- Uses MD5 for hash function, but hashes into $2^{13}$ buckets - can efficiently find collisions
- Results: 10.6 seconds normal, 14.6 seconds under attack
DJBDNS

- Dan Bernstein’s DNS server
- Widely used, intended to be highly secure
  - Dan offers a $500 reward to anyone who finds a security hole
- Code has an explicit check for “hash flooding”: after following a chain for 100 entries, gives up and treats as a cache miss
- So not vulnerable

Perl

- Perl includes hash table implementations as part of the language
- Probably widely used, but is it used for things we care about?
- Proof of concept attack

Perl results

<table>
<thead>
<tr>
<th>File version</th>
<th>Perl 5.6.1 program</th>
<th>Perl 5.8.0 program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl 5.6.1</td>
<td>6506 seconds</td>
<td>&lt;2 seconds</td>
</tr>
<tr>
<td>Perl 5.8.0</td>
<td>&lt;2 seconds</td>
<td>6838 seconds</td>
</tr>
</tbody>
</table>

Table 1: CPU time inserting 90k short attack strings into two versions of Perl.
Bro

- Network Intrusion Detection System
- http://bro-ids.org
- Academic system, not that widely used, but well-respected
- Focus on port scan detector

Port scan detection

- For each source IP, need to track how many distinct destination ports have been contacted
- Bro uses hash table to store <source addr, dest port> pair
- Hash function: XORs them together
- Easy to generate $2^{16}$ input packets that will hash to the same value

Bro results
Fixing the problem

- They focus on better hash functions
  - Make it infeasible to choose x and y such that h(x) = h(y)
  - Cryptographic hash functions
  - Keyed hash functions
  - Universal hashing
- Why is this better than just limiting the length of hash chains?

Universal hashing

- Choose hash function randomly at run time
- Guarantee specific bounds on difficulty of finding a collision
- Random choice of hash function makes it impossible for an attacker to precompute objects that will collide
- Goal is to prevent these types of attacks
- Performs well
Q & A

• What is meant by having “hash tables which may grow large enough to be vulnerable to algorithmic complexity attacks”? Enough buckets or long enough chains?
• Would the break of SHA-1 make it easier for an attacker to create attack inputs for that sort of hash table?
• Why does IPv6 make it easier to exploit this kind of flaw?

Q & A

• How is a universal hash function different from a cryptographic hash? A keyed cryptographic hash?
• Would changing the bucket count help protect against this attack?
• Suppose a hash table had a chaining limit of 100. Wouldn’t a good attack be to sequentially fill every bucket?

Q & A

• Any real world attacks?
  – Linux network routing cache
  – Linux directory entry cache
  – Linux ext2 filesystem
Discussion

• Is this a new idea? Or just new packaging?
  – CLR gives hash attacks as the motivation for universal hashing
  – dbjdns has code in place specifically to protect against these attacks
• What is the contribution?
• What do you need to know to launch one of these attacks?

Discussion

• Are there better ways to defend against these attacks?
• What other types of algorithmic complexity attacks are there?
• Is this an argument against open source software?
• What is the impact of a paper like this? Is Perl going to change hash functions?

Discussion

• Can we use upstream filtering/scanning of the input to protect against these types of attacks?
• Why did they choose Bro to attack?