CS5490/6490 Network Security, Fall 2015
Lecture Notes, November 11th 2015

(Short lecture – Midterm II exams were returned and solutions discussed)

One important problem here is where to fit in the above fields in an IP packet? One possibility is to use the identifier field in the IP header (that is typically used with Flags and offset for fragmentation and reassembly of IP packets).

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Flags</th>
<th>Offset</th>
</tr>
</thead>
</table>

However, the identifier field is only 16 bits long. How do we fit 72 bits of information in 16 bits? First, instead of using two fields for the start address and the end address only one field is used and this field contains the xor of the two addresses (start and end). For example, if a packet flows through routers a, b, c and d, b will insert a ⊕ b, c will store b ⊕ c and d will store c ⊕ d in the xored address field. Now the victim knows d. It can determine c from c ⊕ d, and b from b ⊕ c, etc.

This seems like a good idea but we still need to accommodate the 32-bit address and the distance information in 16 bits. This is achieved by using the following structure.

<table>
<thead>
<tr>
<th>offset</th>
<th>distance</th>
<th>edge fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 bits</td>
<td>5 bits</td>
<td>8 bits</td>
</tr>
</tbody>
</table>

The idea is to divide the 32-bit address into four fragments and randomly include only one fragment at random when a packet is marked. The offset of the fragment is included in the offset field. The distance field is the same as before but has a length of only 5 bits.

Why do they need 3 bits of offset?
In order to make sure the IP addresses obtained from fragmented pieces is correct, a 32-bit hash of each IP address is also computed for each IP address. Hence, an IP address and its hash are equal to 64 bits. This corresponds to eight 8-bit fragments.

There are enough packets are the victim with different fragments that can be put together to obtain the paths to the agents of attack.

Discussion

- Traceback useful only when IP spoofing is used
- Does not stop DDoS attack
- Other approaches being developed to prevent IP spoofing (crypto cookies)
- Incrementally deployable
- A few thousand packets per path construction – computationally demanding
- Requires change in routers
- Only identifies the agent machines

**Botz-for-Sale: Surviving organized DDOS attacks that mimic flash crowds, NSDI 2005**

- Bot – short for robot, process running on a machine that could be used for causing a DoS attack
- Botnets – network of thousands of Bots rented by the hour
- Computer underworld – Massachusetts businessman paid members of the computer underworld to launch DDoS attacks on three competitors.
- To circumvent detection, attackers are increasingly moving away from pure bandwidth floods to stealthy DDoS attacks that masquerade as flash crowds.
- They profile the victim server and mimic legitimate Web traffic browsing behavior of a large number of clients – malicious request differ from legitimate ones in intent but not in content.
- Attack target – CPU resources, disk bandwidth, and database bandwidth
- Many sites do not use passwords to block users. Even if they did, an attack could be launched on the authentication mechanism itself.
- What about giving puzzles that require high processing power? Typically compromised machines have a large amount of spare processing power.
- This paper suggests using graphical puzzles that human clients have to see and solve.

**Bloom Filters:**

A Bloom filter is a method of representing a set \( A = \{a_1, a_2, \ldots, a_n\} \) of ‘n’ elements to support membership queries.

Allocate a vector (array) of \( m \) bits, choose \( K \) independent hash functions, \( h_1, \ldots, h_k \). Here independent hash function mean those that are uncorrelated. For each element, e.g., \( a_1 \), compute \( h_1(a_1), h_2(a_1), h_3(a_1), \ldots, h_k(a_1) \). The hash outputs (1, ..., \( m \)) are positions of the vector of \( m \) bits. The bits corresponding to the positions output by the hash functions are set to 1. We do this for all elements of \( A \).

To test whether ‘\( b \)’ belongs to \( A \) or not, we compute positions \( h_1(b), h_2(b), h_3(b), \ldots, h_k(b) \) and see if get 1’s for these locations. If there is at least one position that is 0 then ‘\( b \)’ does not belong to \( A \). If all the positions have 1s then very likely ‘\( b \)’ belongs to \( A \). However, we cannot be
sure because hash of b might result in positions that are hash of other elements. There is a finite probability by which we can accept an element to belong the set A even though it is not. The probability is called the false positive probability. It can be controlled, by adjusting m and k. When we increase m we reduce collisions and hence reduce the false positive probability. If we increase k, more independent functions are likely to mark different locations. However, increasing k also results in a higher processing overhead.

How are Bloom Filters used in the Botz-for-Sale paper?

A = set of potential problem IP addresses. When a packet arrives, check to see if the IP address belongs to this set. If yes, penalize this IP address. There is some probability by which we might penalize good IP addresses.

In this paper, each bloom filter position is not a single bit but an 8-bit counter. When a packet with IP address IPA is received, all the hash functions $h_1(IP_A)$, $h_2(IP_A)$, $h_3(IP_A)$, .. $h_k(IP_A)$ are computed, and the counters corresponding to these positions are incremented or decremented based on some property of the packet (to be explained later). If the counter reaches a certain threshold a specific action is taken.