Evaluating the Vulnerability of Network Mechanisms to Sophisticated DDoS Attacks

Udi Ben-Porat  
Tel-Aviv University, Israel

Anat Bremler-Barr  
IDC Herzliya, Israel

Hanoch Levy  
ETH Zurich, Switzerland
Why are the systems so vulnerable?

- Computer and network systems have been designed under the assumption that each user aims at maximizing his own performance

- But in "DDoS environment" - some users aim to degrade the performance of other users.

- **Current Status:** Protocols and systems are quite vulnerable in DDOS environment
  - In Crypto terminology we count "Alice" and "Bob" and forgot "Eve" the evil adversary
Distributed Denial of Service (DDoS): Consume the servers/network resources

DDoS: Attacker adds users/queries zombies to flood a server
Sophisticated DDoS Attack

Sophisticated DDoS:

**Goal:** Create maximal damage with limited budget
- **Damage** - Performance degradation
  - Increase errors rate, average delay etc.
- **Budget** - Amount of operations/access allowed

**Motivation:**
- Reduce the (efforts and) cost of the attack
- Make the attack hard to detect
- Allow attacks on limited access servers
Sophisticated Attacks Examples

- **Simple example:** Database server
  - Make hard queries, avoid cache
  - **Goal:** consume CPU time

- **Sophisticated attacks in the research:**
  - **TCP**
    - [TCP] *Low-Rate TCP-Targeted Denial of Service Attacks*
      A. Kuzmanovic and E.W. Knightly
      Sigcomm 2003
  - **Open Hash**
    - [OH] *Denial of Service via Algorithmic Complexity Attacks*
      Scott A. Crosby and Dan S. Wallach
      Usenix 2003
  - **Admission Control**
    - [RoQ] *Reduction of Quality (RoQ) Attacks on Internet End-Systems*
      Mina Guirguis, Azer Bestavros, Ibrahim Matta and Yuting Zhang
      Infocom 2005
Study Objective

- Propose a DDoS Vulnerability performance metric
  - Vulnerability Measure
  - To be used in addition to traditional system performance metrics
- Understanding the vulnerability of different systems to sophisticated attacks

This Talk

- Describe DDoS Vulnerability performance metric
- Demonstrate metric impact
  - Hash Table: Very Common in networking
  - Performance (traditional): OPEN equivalent CLOSED
  - Vulnerability analysis: OPEN is better than CLOSED!!
Distributed Denial of Service (DDoS)

- Attacker adds more regular users 😊
- Loading the server - degrading server performance
Sophisticated DDoS

- Attacker adds sophisticated malicious users 😈
- Each user creates maximal damage (per attack budget)

Attacker

![Diagram showing attacker, malicious users, server, and server performance with bars for DDoS, Normal, and S. DDoS]

U. Ben-Porat / A. Bremler / H. Levy
Infocom 2008
8
Vulnerability Factor Definition

\[
\text{Vulnerability}(\text{Cost } = c) = \max_{st} \frac{\Delta\text{Performance}(\text{Malicious}_{st}, c)}{\Delta\text{Performance}(\text{Regular}, c)}
\]

\(\text{Vulnerability} = v\) means: In terms of performance degradation 1 malicious user equals \(v\) regular users

(st = Malicious Strategy)  U. Ben-Porat / A. Bremler / H. Levy
Infocom 2008
Vulnerability Factor Definition

\[ \text{Vulnerability}(\text{Cost} = c) = \frac{\Delta \text{Performance}(\text{Malicious}, c)}{\Delta \text{Performance}(\text{Regular}, c)} \]
Vulnerability Factor - example

Example: vulnerability of server serving N users:

<table>
<thead>
<tr>
<th>Attack Type:</th>
<th>Add k regular users</th>
<th>Add k malicious users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Degradation:</td>
<td>5%</td>
<td>20%</td>
</tr>
</tbody>
</table>

Vulnerability(k): = \( \frac{20\%}{5\%} = 4 \)

When allowing \( k \) additional users to this system (in an hostile environment), we should be prepared for performance degradation 4 times worse than the expected.
Demonstration of Vulnerability metric: Attack on Hash Tables

- Hash table is a data structure based on Hash function and an array of buckets.
- Central component in networks

Operations: Insert, Search and Delete of elements according to their keys.

Insert (element) 

<table>
<thead>
<tr>
<th>key</th>
</tr>
</thead>
</table>

Hash(key) 

User 

Server 

Buckets
Hash Tables

**Open Hash**

- Bucket = list of elements that were hashed to that bucket
- Insertion verifies the element does not already exist

**Closed Hash**

- Bucket = one element
- Collision -> the array is repeatedly probed until an empty bucket is found
Vulnerability: OPEN vs. CLOSED

Performance metrics: OPEN = CLOSED*
What about Vulnerability? OPEN = CLOSED ?
(* when the buckets array of closed hash is twice bigger)

Open vs. Closed

- **In Attack vulnerability:**
  - While attack is on: Attacker’s operations are CPU intensive \(\rightarrow\) CPU loaded

- **Post Attack vulnerability :**
  - Loaded Table\(\rightarrow\) insert/delete/search op’s suffer
Attacker strategy (InsStrategy)

Strategy:
- Insert \( k \) elements (cost=budget=\( k \)) where all elements hash into the same bucket
- Theorem: InsStrategy is Optimal
  - For both In Attack and Post Attack

Attack Results

Open Hash:
One long list of elements

Closed Hash:
Cluster
Attack effects: Open vs Closed

- **Open Hash** - One long list of elements
- **Closed Hash** - Cluster
Analytic Results

- We compare the vulnerability of the Open and Closed Hash using analysis of the Vulnerability (probability + combinatorial analysis).

- Common approach: Open Hash table with \( M \) buckets is performance-wise equivalent to a Closed Hash table with \( 2M \) buckets.

- We present graph: \( M = 500 \) and \( k = N \), i.e.,
  - The additional users \( k \) double the number of values in the table.
In every malicious insertion, the server has to traverse all previous inserted elements (+ some existing elements)

\[ V \sim K/2 \]

**Open Hash**

\[ V = \left( \sum_{i=1}^{k} \frac{i}{M} \cdot \left( \frac{N}{i} \right) \cdot \left( \frac{M-N}{i} \right) \cdot \left( \frac{1}{i} \right) \right) / \left( \sum_{i=1}^{k} \frac{1}{1-(\frac{1}{M+i})} \right) \]

**Closed Hash**

\[ V = \frac{k+2L+1}{M+2L+2} \]
**Post Attack: Operation Complexity**

**Open Hash**

Vulnerability = 1

No Post Attack degradation in Open Hash

(Only small chance to traverse the malicious list)

**Closed Hash**

Big chance the operation has to traverse part of the big cluster

\[
\Delta V_{CH}(k) = \left( E[Q_M] - \frac{1}{\theta} \right) \left( \frac{1}{1-(k+N)/M} - \frac{1}{1-N/M} \right),
\]

where \( E[Q_M] = \sum_{0 \leq h \leq N} \binom{N}{h} \binom{N-h}{k+2}^{-1} \left( \frac{c(c+1)}{2M} + 1 - \frac{c}{M} \right)^{-1} \).
Post Attack Operation Complexity

- **Open Hash**

  This performance parameter is not vulnerable to sophisticated attacks:

  The complexity is always the average list length $(L+k/M+1)$ regardless of the positions of the $k$ elements.

- **Closed Hash - Cluster**

  **Regular:**

  $$\frac{1}{1 - (\alpha + k/M)}$$

  **Malicious:** \(\sim k^2k/2M\)

  Every insert element which is mapped into a bucket in the Cluster (not just specifically the bucket the attackers used) will need to search from its position to the end of the cluster.
Queueing - FCFS

Consider $M/G/1$ model with arrival rate $\lambda$ and service times distributed as a random variable $X$. The expected waiting time of an arbitrary arrival:

$$\text{Average Waiting Time} = \frac{\lambda E(X^2)}{2(1 - \lambda E(X))}$$

- **Regular Users**: Rate $\lambda_1$, job size distributed like $X$

- **Attackers**: Send jobs of size $KE(X)$ at rate $\lambda_2 = \lambda_1 / K$ where $K$ parameter of the attack (this increases the second moment of $X$)

- **Budget**: Both users use the same budget $= \lambda_1 E(X)$

- **Vulnerability**: Unbounded $\sim k$
  (The second moment of $X$ plays an important role).
**Post Attack: account for queuing**

- **Queue** holds the requests for the server
- **Waiting Time** = \( \frac{\lambda E(X^2)}{2(1-\lambda E(X))} \) (\( X = \) service time)
- **Vulnerability of the (post attack) Waiting Time?**

![Diagram of queuing with server and request table]
**Post Attack Waiting Time**

**Open Hash:**
- **Vulnerable!!** While in the model of *Post Attack Operation Complexity* the Open Hash is not Vulnerable!

\[ V_{OH}(k) = 1 + \frac{(1 - \lambda L)(k - 1)(1 - \frac{1}{M})}{(1 - \lambda L)(k - 1)(\frac{1}{M}) + 2L + 1 - \lambda L(L + \frac{1}{M})} \]

**Closed Hash:**
- Drastically more vulnerable: Clusters increase the *second moment* of the service time

- Queue becomes unstable
Stability Point

- For N above this **stability point**, the arrival rate of regular user operations is greater than the expected service rate.

- For specific Hash state we can calculate the maximum magnitude of an attack so that the Hash remains stable after the attack has ended.
In hostile environments one can no longer follow the simple approach of relying only on complexity based rules of operations in order to comply with the performance requirements of a system.

For example, based on traditional performance → double the size of the Closed Hash table (rehashing) when the table reaches 70/80% load.

In a hostile environment the rehashing must be done much earlier (in the previous example at 48% load the Closed Hash is no longer stable).
Conclusions

- Closed Hash is much more vulnerable to sophisticated DDoS than Open Hash
  - In contrast: the two systems are considered equivalent via traditional performance evaluation.

- After attack ends, regular users still suffer from performance degradation

- Application using Hash in the Internet, where there is a queue before the hash, has high vulnerability.
Related Work

- **[RoQ]** Attack measurement: Potency
  - Potency measure specific attack $\iff$ Vulnerability measures the system
  - Meaningless without additional numbers $\iff$ Vulnerability is meaningful comparable information based on this number alone

- **[OH]** Studies attack only on Open Hash.
  - Our work is comparing Closed to Open Hash, also analyzing the post attack performance degradation
Questions ?