Skiplist Timing Attack Vulnerability

Eyal Nussbaum
PhD Student, Communication Systems Engineering
School of Electrical and Computer Engineering
Ben-Gurion University of the Negev
Advisor: Professor Michael Segal
Talk Overview

- Introduction
- Probabilistic Skiplist
- Skiplist structure mapping
- Possible attacks on Skiplists
- Splay List as a proposed defense
- Summary
Introduction

• Database Characteristics:
  ▫ Underlying data structure – graphs, trees, lists and so on.
  ▫ Data types/formats – text, discrete or continuous numeric values, coordinates and others.
  ▫ Query types and behavior.

• Targets
  ▫ Identify potential weaknesses and attack vectors based on DB characteristics, and offer defenses.
  ▫ Offer computational complexity for attack/defense.
Run-time Based Attack

• The underlying architecture of a database may be comprised of a single or multiple data structures: graphs, trees, stacks, etc...
• The organization of the data may hold information regarding the data itself (as in the case of a binary search tree).
• Run-time of queries is also dependent on the structure and may leak information
  ▫ Futoransky et al. describe such an attack on SQL databases (insertion attack).
• We show an example of an attack based on the Skiplist structure.
  ▫ Skiplists are a probabilistic alternative to balanced trees.
  ▫ Maintain an ordered structure with multiple levels.
  ▫ Contains $log \ n$ levels with $\frac{n}{2^{l-1}}$ nodes per level $l$. 
Proababilistic Skiplist - Example

Figure 1 – 4 level Skiplist with 15 nodes

- **Skiplist creation:**
  - Search for ordered placement of node.
  - Insert node at level 1.
  - With 0.5 probability, add next level to node.
  - Continue to subsequent level with probability 0.5 until either next level was not added, or max level has been reached.

- **Skiplist implementations:**
  - MemSQL, Redis
We give an algorithm, SkipListMap, that maps the structure of a given probabilistic Skiplist using the search function.
  ▫ The size of the structure, $n$, is known.
  ▫ The structure holds unique values.
  ▫ The range of possible values in the structure is known and is of size $O(n)$.
  ▫ The runtime of the search algorithm is consistent.

Using SkipListMap to discover the structure of the Skiplist allows us to perform attacks.

Goal:
  ❑ Restructure the Skiplist to cause worst case performance.
  ❑ Create hidden channel between two parties.
SkipListMap Algorithm

- Consists of two phases:
  - Search time mapping
  - Skiplist reconstruction
- Search operation example for Skiplist in figure 2
  - Search for “10” – requires 6 comparisons.

Figure 2 – Skiplist search example
SkipListMap - Search Time Mapping

**Algorithm 1** Map Skiplist search times

1: **procedure** MapRuntime(skiplist, values)
2:    runtimes = new Array
3:    for $x_i$ in values do
4:        $T_{x_i} \leftarrow$ runtime of skiplist.find($x_i$)
5:        runtimes.append($T_{x_i}$)
6:    end for
7:    $T_{min} \leftarrow \min($runtimes$)$  \quad \triangleright \text{Minimum over all runtimes}
8:    for $T_{x_i}$ in runtimes do
9:        $T_{x_i} = \frac{T_{x_i}}{T_{min}}$
10:   end for
11:   return runtimes
12: **end procedure**

- Search for all possible values, $x_i$, in the Skiplist.
- For each value found, denote its search time $T_{x_i}$.
- Denote the the lowest runtime to be $T_{min}$.
- Normalize runtimes based on $T_{min}$ such that $T_{min} = 1$.
- Normalized $T_{x_i}$ is the length of the search path to $x_i$. 
Search Time Mapping - Example

• For our example:
  - $T_1 = 3$, $T_2 = 2$, $T_3 = 5$, $T_4 = 3$,
  - $T_5 = 6$, $T_6 = 5$, $T_7 = 1$, $T_8 = 5$,
  - $T_9 = 4$, $T_{10} = 6$, $T_{11} = 7$, $T_{12} = 2$,
  - $T_{13} = 6$, $T_{14} = 7$, $T_{15} = 5$. 
Create empty SkipList with \( \log n \) levels (in our example, 4)

Insert nodes in order of increasing values of \( x_i \), beginning at level 1

- After each level insertion attempt, search for \( x_i \).
- Repeat until correct search time is found.
Reconstruction - Example

- Reconstruction of first 4 nodes.
- Note that once a node level is chosen, inserting nodes to the right does not change search time of previous nodes.

![Diagram showing reconstruction process](image-url)
Skiplist Runtime Attack

- Runtime Attack requires “write” access
- Restructure the Skiplist to cause worst case performance.
- Remove all items which exists above level 1.
- Re-insert all items that were removed. Approximately 0.75 will be in level 1.
- Repeat removal/insertion until reducing Skiplist structure to a linked list with a search time of $O(n)$. 
Skiplist Hidden Channel Attack

- Hidden Channel Attack requires 2 parties, one with “write” access.
  - Transmitter and Receiver
- Original Skiplist database is distributed publicly.
- Each attacker maps the Skiplist structure.
- Transmitter holds private knowledge regarding nodes.
- Transmitter selectively removes and re-inserts nodes, marking them.
  - Allows transfer of 1 extra bit of information regarding nodes.
    - For example – gender information, placebo/drug differentiation...
  - Alternatively, allows $n$-bit message encoding.
- Transmitter re-distributes Skiplist with structure change only.
- Receiver can decipher hidden channel using SkiplistMap.
Splay List: Skiplist Variation

• Suggested defense from SkipListMap attacks – conceal runtime.
• Propose Splay List structure, a variant of Skiplist.
• Based on Splay Tree concept of re-ordering nodes when search is performed.
• Splay algorithm (after the search has been completed)
  ▫ Swap levels between 2 nodes: random and searched.
  ▫ Remove connections when lowering level, connecting preceding and succeeding nodes.
  ▫ Add connections when increasing levels, disconnecting preceding and succeeding nodes.
• Runtime is $O(\log n)$
Splay List Behavior

• Addition and removal of nodes remains the same as Skiplist.
• Change in the search function:
  ▫ Denote the corresponding searched node $u_x$.
  ▫ Select a random node $u_r$.
  ▫ Swap between the levels of $u_x$ and $u_r$ using the Splay Node algorithm.
• Slightly increasing the runtime of the search but remaining in $O(\log n)$.
  ▫ Search for additional node
  ▫ Lowering level of node - similar to node removal
  ▫ Raising level of node - similar to node addition
Algorithm 3 Splay Skiplist node: change node level

1: procedure SPLAYNODE(v, newLevel, prevNodesArray)
2: \( max \leftarrow \text{maxLevel}(v) \) \( \triangleright \) Find current max level of \( v \)
3: if \( max > newLevel \) then
4: \( \text{for } l \text{ in } max + 1 \ldots newLevel \) do
5: \( v_l \leftarrow \text{newnodeLevel}(v) \) \( \triangleright \) Create new level in node \( v \)
6: \( v_l.next \leftarrow \text{nextNodesArray}[l].next \) \( \triangleright \) Connect \( v \) to next node in level
7: \( \text{prevNodesArray}[l].next \leftarrow v_l \) \( \triangleright \) Connect \( v \) to previous node in level
8: end for
9: end if
10: if \( max < newLevel \) then
11: \( \text{for } l \text{ in } newLevel + 1 \ldots max \) do
12: \( \text{prevNodesArray}[l].next \leftarrow v_l.next \)
13: \( v.deleteLevel(l) \) \( \triangleright \) Remove level \( l \) from \( v \)
14: end for
15: end if
16: end procedure
Figure 3 (Splay List Search)

Search for node in Splay List.

Node 9 found.

Node 4 chosen for swap and found in Splay List.

Top levels swapped between nodes 9 and 4.
Summary

• Probabilistic Skiplist structure to be vulnerable to a timing attack.
  ▫ Allows mapping of the structure.
• Possible attacks:
  ▫ Runtime attack – performance degradation.
  ▫ Hidden Channel attack – undetected transfer of data using structure.
• Proposed defense – Splay list.
  ▫ Randomize structure after search.
  ▫ Retain $O(\log n)$ performance.
• Future directions:
  ▫ Consider the behavior of multiple releases over time.
  ▫ Consider attacks based on other data structures (trees, graphs, etc...)

Summary
Thank You!