Vulnerability of Network Mechanisms to Sophisticated DDoS Attacks

Udi Ben-Porat, *Student Member, IEEE,* Anat Bremler-Barr, *Member, IEEE,* and Hanoch Levy, *Member, IEEE.*

Abstract—In recent years we have experienced a wave of DDoS attacks threatening the welfare of the internet. These are launched by *malicious* users whose only incentive is to *degrade the performance of other, innocent, users*. The traditional systems turn out to be quite vulnerable to these attacks.
The objective of this work is to take a first step to close this *fundamental gap*, aiming at laying a foundation that can be used in future computer/network designs taking into account the malicious users. Our approach is based on proposing a metric that evaluates the vulnerability of a system. We then use our vulnerability metric to evaluate a data structure which is commonly used in network mechanisms – the Hash table data structure. We show that Closed Hash is much more vulnerable to DDoS attacks than Open Hash, even though the two systems are considered to be equivalent by traditional performance evaluation. We also apply the metric to queueing mechanisms common to computer and communications systems. Further more, we apply it to the practical case of a hash table whose requests are controlled by a queue, showing that even after the *attack has ended*, the regular users still suffer from performance degradation or even a total denial of service.

Index Terms—DDoS, hash, queue, vulnerability, metric, malicious.

1 Introduction

In recent years the welfare of the Internet has been threatened by malicious Distributed Denial of Service (DDoS) attacks. DDoS attackers consume the resources of the victim, a server or a network, causing a performance degradation or even the total failure of the victim. In DDoS attacks, attackers send a large amount of traffic, consuming the CPU or the bandwidth of the victim. The ease with which such attacks are generated makes them very popular. According to Labovitz et al. [1], large backbone networks observe DDoS attacks on a daily basis.

The basic form of a DDoS attack is the *simple high-bandwidth DDoS attack*. That is, simple brute force flooding, where the attacker sends as much traffic as he can to consume the network resources, namely the bandwidth of the server's incoming link, without using any knowledge of the system design. This can be achieved by sending a huge amount of traffic over a raw socket or by using an army of Zombies[1], each mimicking the requests of a regular user.

Sophisticated low-bandwidth DDoS attacks use less traffic and increase their effectiveness by aiming at a weak point in the victim's system design, i.e. the attacker sends traffic consisting of complicated requests to the system. While it requires more sophistication and understanding of the attacked system, a low bandwidth DDoS attack has three major advantages in comparison to a high bandwidth attack: 1. Lower cost – since it uses less traffic; 2. Smaller footprint – hence it is harder to detect; 3. Ability to hurt systems which are protected by flow control mechanisms. An example of such attacks is an attack against HTTP servers by requesting pages that are rarely requested (forcing the server to search in the disk). Similar attacks can be conducted on search engines or on database servers by sending difficult queries that force them to spend much CPU time or disk access time. In fact, an example of such a sophisticated attack can be seen even in the classic SYN attack [2] which aims hurting the TCP stack mechanism at its weakest point, the three-way-handshake process and its corresponding queue.

In this work we focus on sophisticated low-bandwidth attacks. We define *sophisticated low-bandwidth DDoS attacks* as attacks that increase their effectiveness by aiming at hurting a weak point in the victim's system design, i.e. the attacker sends traffic consisting of complicated requests to the system. This generalized definition covers all the different attack types described in recent papers [3]–[9] including the two new major breeds: the Complexity Attack [3] and the Reduction of Quality (RoQ) attacks [5], [6]. In Section 2 we give a short overview of the different DDoS attack types that fall into the category of sophisticated attacks.

Our first contribution in this work (Section 3) is a proposal of a new metric that evaluates the vulnerability of a system for any kind of sophisticated attacks. Our approach is based on proposing a **Vulnerability**

U. Ben-Porat is with the Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland. e-mail: ehud.benporat@tik.ee.ethz.ch.
Anat Bremler-Barr is with the Computer Science Dpt., Interdisciplinary Center, Herzliya, Israel. email: bremler@idc.ac.il.
Hanoch Levy is with the Computer Science Dpt., Tel-Aviv University, Tel-Aviv, Israel. email: hanoch@post.tau.ac.il.

1. A zombie is a compromised machine that is controlled by a remote hacker. The current estimate is that there are millions of zombie machines today, in private homes and institutes.

Metric that accounts for the maximal performance degradation (damage) that sophisticated malicious users can inflict on the system using a specific amount of resources (cost) normalized by the performance degradation attributed to regular users, using the same resources. Having a metric for computer security [10] is an important step for improving security. As a well known phrase states : "If you cannot measure, you cannot manage. If you cannot manage, you cannot improve". Our metric provides the ability to understand the resilience of systems to DDoS attacks and, in the case that multiple algorithms/designs are available for the same problem, helps in selecting the most resilient one.

Vulnerabilities to sophisticated DDoS attacks seem to be an inherent part of existing computer and network systems. Traditionally, these systems have been designed and operated under the underlying fundamental principle that *each user aims at maximizing the performance he/she receives from the system*. While operational strategies have been divided to *social optimization* (optimize operation to the benefit of the overall population) or *individual optimization* (each user acts to improve his own performance), see e.g. [11], the basic paradigms were still based on this principle, namely that each user aims at maximizing his own performance. This paradigm does not hold any more in the new "DDoS environment" where some users do not aim to maximize their own performance, but rather to degrade the performance of other users. Thus, there is a *major gap* between traditional assumptions on user behavior and practical behavior in modern networks. Due to this gap it is both natural and inherent that traditional algorithms, protocols and mechanisms will be quite vulnerable in this environment.

Our second contribution, presented in Section 4, is an evaluation of the **Hash data structure**, commonly used in network mechanisms, using our vulnerability metric. The sophisticated attack against the Hash data structure was first introduced in the work of Crosby and Wallach [3]. However, their work focused only on Open Hash, using only experimental results on real-life Hash implementations. In contrast, we provide analytic results using our *Vulnerability Factor* metric on the two Hash types: Open Hash and Closed Hash. An important finding of this simple analysis, which can affect practical designs, is that these two common Hash types (which are known to be equivalent according to many performance aspects) drastically differ from each other in their vulnerability to sophisticated attacks.

Our third contribution, presented in Section 5, is an examination of the vulnerability of some queueing mechanisms, commonly used in computer networks. We address this subject by considering a very simple example of the First Come First Served (FCFS) queue. We show that though attackers, under this mechanism, "must wait like everyone else" and thus have no control of their place in the queue, they can still cause significant damage even if they use, on average, the same amount of resources that regular users do. The strategy of the malicious users is simple - just submit *large jobs* from time to time. In fact the vulnerability level (using the proposed metric) of this system can be unbounded.

Our fourth contribution, provided in Section 6, shows that in the case of an attack either on an Open Hash system or on a Closed Hash system, the regular user still suffers from performance degradation or even a total denial of service, even **after the attack has ended** (note that the degradation is not due to the extra load of the attack but due to its sophistication). Our results show that even a data structure whose amortized complexity (whether attacked or not) is $O(1)$, such as Open Hash, is vulnerable to malicious attacks when the structure is combined with a queue. Although the data structure itself is immune to complexity attacks (since it still operates with $O(1)$ amortized complexity), the waiting time in the queue can dramatically increase if an attacker manages to increase the variance of the request processing time.

As far as we are aware, we are the first to point out *post attack* performance degradation. We demonstrate this by using the analysis of a practical and common case that combines the Hash table data structure with a queue preceding it (post-attack degradation is demonstrated, for Closed Hash, also in Section 4). Using this model, we further derive the precise size of the attack that will "harm" the Hash system in such a way that, even after the attack has ended, it will make the regular users suffer from complete *denial of service* (and not only from performance degradation). In Section 7 we compare the vulnerability of Open and Closed Hash. A short paper with preliminary results from this work has been published in [12].

2 Sophisticated Attacks

We define a *Sophisticated DDoS attack* as an attack in which the attacker sends traffic aiming to hurt a weak point in the system design in order to conduct denial of service or just to significantly degrade the performance (such as Reduction of Quality attacks like [5], [6]). The sophistication lies in the fact that the attack is tailored to the system design, aiming to increase the effectiveness of the attack.

The motivation of the attacker is to minimize the amount of traffic it sends while achieving similar or even better results. Using sophisticated attacks the attacker reduces the cost of attacks i.e., reduces the number of required zombies[2] and reduces the sophistication in coordinating the attack. Moreover, the

2. Today, there is a market in zombie machines. Attackers can skip the time consuming process of taking control of new machines and just buy "off the shelf" zombies. Hence the number of machines required to launch the attack translate directly to the cost of money.

use of sophisticated attacks increases the likelihood that the attack will succeed in going unnoticed (going under the radar) by the DDoS mitigation mechanisms, which are usually based on detecting the fact that the victim was subject to higher-than-normal traffic volume. Note that designing a sophisticated attack tool requires only a one-time effort of understanding the system design in order to find its weak point. However, after such a tool is constructed, other attackers can use the tool as an "off-the shelf" black box attack tool without any knowledge of the system. Similar evolutions were observed in other DDoS attack types such as "SYN-attack" where general tools such as Trinoo and TFN were designed and distributed over the web to layman attackers [13]. Due to the complexity of the applications, they are usually more vulnerable to Sophisticated DDoS attacks. However, the Sophisticated attacks are not just against applications and may also be conducted against protocols (see, for example, attacks against TCP [7]).

Roughly speaking, we can classify the methods of launching sophisticated attacks into two classes: *Malicious Job Content* and *Malicious Jobs Arrival Pattern*.

1) **Malicious Job Content** - Attacker exploits the worst-case performance of the system which differs from the average case that the system was designed for. In [3] Crosby and Wallach demonstrate the phenomenon on Open Hash table implementations in two versions of Perl, the Squid web proxy, and the Bro intrusion detection system. They showed that an attacker can design an attack that achieves worst case complexity of $O(n)$ elementary operations per insert operation instead of the average case complexity of $O(1)$. It caused, for example, the Bro server to drop 71% of the traffic. Examples of systems and algorithms that can potentially yield to complexity attacks are quicksort [14] regular expression matcher [15], Intrusion Detection Systems [9], [16] and the Linux route-table cache [17]. Moscibroda and Multu [18] discuss a sophisticated DDoS attack which degrades the performance of a Multi-Core System. The attack is conducted by one process (running on one core) targeting the weakest point in such a system - the shared memory - and degrades the performance of other processes in the system. Smith et al. [9] describe a low bandwidth sophisticated attack on a Network Intrusion Detection System (NIDS). This attack disables the NIDS of the network by exploiting the behavior of the rule matching mechanism by sending packets which require a very long inspection times. Another example is the SSL handshake dropping DoS attack [19], which abuses the fact that clients can request the server to perform computationally expensive operations without doing any work themselves. In the 802.11 standard, malicious users can steal additional time share by using Network Allocation Vectors to reserve a long period of time for themselves [20]. Similar attacks can be conducted against cellular networks; In [21] the authors show how malicious users can behave according to the worst case scenario of the retransmission algorithm to obtain a larger time share.

2) **Malicious Jobs Arrival Pattern** - Attacker exploits the (stochastic) worst case traffic pattern that can be applied to the system. This case is similar to the first one with the difference that the worst case scenario involves a specific traffic arrival pattern of requests from multiple users. This type of attack is demonstrated in the Reduction of Quality (RoQ) attacks papers [5], [6], [9]. RoQ attacks target the adaptation mechanisms by hindering the adaptive component from converging to steady-state. This is done by sending - from time to time - a very short duration of surge demand imitating many users and thus pushing the system into an overload condition. Using a similar technique, Kuzmanovic and Knightly [7] presented the Shrew Attack which is tailored and designed to exploit TCP's deterministic retransmission timeout mechanism. In [22] the authors show how a group of malicious users can attack a scheduler of a cellular network and increase their allocated time share significantly by coordinating their reported channel condition values. The pattern created by this behavior allows them to consistently obtain exaggerated priority and obtain a larger time share (at the expense of others). Another example of an attack exploiting the stochastic worst case is given in Bremler-Barr et al. [4]. There it is shown that Weighted Fair Queueing (WFQ), a commonly deployed mechanism to protect traffic from DDoS attacks, is ineffective in an environment consisting of bursting applications such as the Web client application.

3 The Vulnerability Factor

Our work is based on the observation that the traditional performance analysis of computer systems and algorithms is not adequate to analyze the vulnerability of a system to sophisticated DDoS attack and a new metric is needed. Such a vulnerability metric can help a system designer to:

Choose between two equal systems: Such universal vulnerability metric can be used to compare the vulnerability of two alternative systems. For example, a system designer of a modern cellular base station could use the vulnerability metric to compare between the CDF [22] and PFS [21] wireless scheduling algorithms.

Select the values of system operating variables: The analysis (either analytically or empirically) of the vulnerability value can expose the contribution of the different system parameters to the system vulnerability and help the designer to balance trade-offs between the system vulnerability and its efficiency. For example, in the PFS wireless scheduler [21], increasing the maximal number of retransmissions increases the base station throughput on the one hand while increasing its vulnerability on the other hand.

Use vulnerable systems safely: A vulnerability metric assessing the amount of damage that can be inflicted by malicious sophisticated users can help to use known vulnerable systems wisely. For example, consider the Bro intrusion detection systems that can drop up to 71% of the traffic during an attack [3]. A company that already owns and uses vulnerable Bro systems can still use them safely by rearranging the servers and the network such that none of them will have to handle more that 29% of its original traffic (and hence will not drop traffic even under attack).

It is important to note that the analysis of the worst-case performance of a system versus that of the average-case is not sufficient in order to comprehend the vulnerability of a system. This is due to three major factors: First, the worst case analysis does not take into consideration the cost of producing the worst case scenario. However, in the context of analyzing the vulnerability of a system, the cost is an important factor, since a scenario that is expensive to produce, does not really pose a threat to the system. Second, traditionally the assumption in the worst case analysis of a system is that all the participants follow the protocol rules. However, dealing with a malicious attacker, the attacker can form his own protocol, mainly designed to damage the system as much as possible. Third, usually the worst case analysis deals with the most difficult input into the system per user. However in the vulnerability assessment we also need to take into account the most difficult scenario of interaction between the users of the system. That is, we also need to analyze the system using the "stochastic" worst case scenario (as shown in the sophisticated attacks [5], [7]).

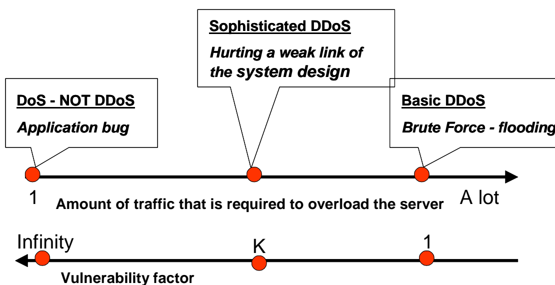


Fig. 1. Vulnerability Factor and Sophisticated DDoS attack.

Our proposal for the definition of the Vulnerability Factor of a system is to account for the maximal performance degradation (damage) that sophisticated malicious users can inflict on the system using a specific amount of resources (cost) normalized by the performance degradation attributed to regular users using the same amount of resources. Figure 1 demonstrates the concept of the Vulnerability Factor. Mechanisms that are vulnerable only to brute force attacks will have a Vulnerability Factor of one, while mechanisms that are vulnerable to sophisticated attacks will receive a factor of $K > 1$ indicating that the maximal attacker's power is equivalent to that of K regular users[3].

Formally, we define the Vulnerability Factor as follows: Let the $usersType$ parameter be equal to either regular users (R) or malicious attackers (M_{st}) with strategy st. Note that we use the plural terms since some of the attack types occur only in specific scenarios of multiple coordinated access of users to the system [5]. Let the $budget$ be the amount of resources that the users of $usersType$ spend on producing the additional traffic to the system, i.e., the cost of producing the attack is limited by the $budget$ parameter. The $budget$ can be measured differently in the various models, e.g. as the required bandwidth, or the number of required computers or as the required amount of CPU and so on. Let $\Delta Perf(usersType, budget)$ be the change in the performance of the system due to being subject to additional traffic added to the regular traffic of the system, where the traffic is generated by users from type $usersType$ with resource $budget$. The performance can be measured by different means such as the CPU consumption, the delay experienced by a regular user, the number of users the system can handle, and so on.

We define the *Effectiveness* of a strategy st on the system as

$$E_{st}(budget = b) = \frac{\Delta Perf(M_{st}, b)}{\Delta Perf(R, b)}, \tag{1}$$

and the *Vulnerability Factor V* of the system as:

$$V(budget = b) = max_{st}\{E_{st}(b)\}. \tag{2}$$

If the *Vulnerability Factor* of a system equals the effectiveness of a strategy st, then st is said to be an *Optimal Malicious Users Strategy*. In practice, there may be a system in which no strategy can be proved to be the optimal because the effectiveness of all other possible strategies cannot be evaluated or bounded. In this case, the most effective (known) malicious strategy is considered to be a lower bound on the vulnerability of the system. Note that for the same system there can be more than one *Optimal Malicious Users Strategy* and that an *Optimal Malicious Users Strategy* maximizes $\Delta Perf(M_{st}, b)$. Observe that for systems invulnerable to attacks $V = 1$ since $\Delta Perf(M_{st}, b) = \Delta Perf(R, b)$.

Note that we focus on the attacker's strategy that has the maximum effect in performance degradation, since the common assumption in the field of security is that the vulnerability of a system is measured at its weakest point.

3. Placing the DoS attack on the same baseline, the DoS attack (exploiting a bug in the application, leading to a total system failure) receives the factor of infinity.

3.1 Related Work

The first step for measuring system vulnerability, was done by Guirguis et al. [5][4], which concentrated only on measuring RoQ attacks. Their work proposes the attack *Potency* measure which evaluates the amount of performance degradation inflicted by an attacker according to his budget. The reader can observe that this measure is similar to our definition of $\Delta Perf(M_{st}, budget = b)$. However, the difference between the *Effectiveness* (E_{st}) and the *Potency* measures, is that we normalize the performance degradation the system suffers due to an attacker by the performance degradation the system suffers due to a regular user. Clearly, *Potency* can be generalized to measure any sophisticated attack, however we think that the *Effectiveness* is preferable since it allows for deriving meaningful information based on this (dimensionless) number alone. Consider the case of a system with a Vulnerability Factor of K. In this case we can deduce that the system can cope with a number of sophisticated attackers, which is only $\frac{1}{K}$ of the number of regular users the system can handle. In contrast, if the potency level is P it means that the attacker obtains P units of damage per one unit of cost; this number is meaningless without comparing it to the potency of other scenarios, such as the potency of a regular user, or the potency of other attacks. Moreover, the unit used by the potency measure (damage per cost) is not fully defined and left for the interpretation of whoever uses it. Both the *Potency* and *Effectiveness* measures focus on the power of a specific attack. In contrast, we propose that a more interesting metric is the *Vulnerability Factor* that focuses on the *system* and provides bounds on its performance under any type of attack.

In this work we present results relating to the vulnerability of Hash systems. The sophisticated attack against Hash tables was first introduced in [3] which concentrated only on Open Hash, using only experimental results on real-life Hash implementations. In contrast, we provide analytic results using our *Vulnerability Factor* metric on the two Hash types: Open Hash and Closed Hash. An important finding of this simple analysis, which can affect practical designs, is that these two common Hash types (which are known to be equivalent according to many performance aspects) drastically differ from each other in their vulnerability to sophisticated attacks. Moreover, by analyzing the effect of sophisticated attacks on queueing with Hash, our work further shows that the regular users suffer from performance degradation even after the attack has ended.

4. The primal work of Gilgor [23] presents a formal specification and verification in order to prevent denial of service, but did not try to measure the vulnerability of the system. Moreover, the work concentrates on denial of service of a different type, where some users can lead the system to a situation where part of the users reach a "deadlock" situation.

4 Sophisticated Attacks on Hash Tables

Many network applications, such as the Squid Web Proxy and the Bro Intrusion Detection System, use the common Hash data structure. This data structure supports the dictionary operations of Insert, Search and Delete of elements according to their keys. Hash tables are used when the number of keys (elements) actually stored is relatively low compared to the total number of possible key values. A Hash table is based on an array of size M, where an entry in the array is called a *bucket*. The main idea of a Hash table is to transform the key k of an element, using a Hash function h, into an integer that is used as an index to locate a bucket (corresponding to the element) in the array. In the event that two or more keys are hashed to the same bucket, a collision occurs and there are two Hash strategies to handle the collision: Open Hash (a.k.a Closed Addressing) and Closed Hash (a.k.a Open Addressing). In Open Hash each bucket in the array points to a linked list of the records that were hashed to that bucket. In Closed Hash all elements are stored in the array itself as follows; In the Insert operation, the array is repeatedly probed until an empty bucket is found. The Hash function in this case can be thought of as being extended to be a two parameter function depending on the key k and the number of the probe attempts i i.e. $h(k, i)$ (for detailed explanations about Open and Closed Hash see [24]). The particular probing mechanism where $h(k, i) = H(k) + i - 1 (mod M)$ is called *linear probing*.

In the Hash data structure (Closed or Open), the average complexity of performing a Hash operation (Insert, Search and Delete) is $O(1)$ elementary operations[5] while the worst case complexity is $O(N)$ elementary operations per Hash operation, where N is the number of existing elements in the Hash table. The Insert operation consists of a search to check whether the element already exists in the Hash table and in addition, an insertion of the new element (in the case where it does not exist). In both Closed and Open Hash the Insert operation of a new element is the most time consuming operation.

In [3] it is shown that an attacker, in the common case where the Hash function is publicly known[6], can drive the system into the worst case performance, and thus dramatically degrade the performance of the server application. Naturally, the Hash example belongs to the class of **Malicious Job Content** sophisticated attacks. Note that Universal Hash functions were introduced by Carter and Wegman [25] and are cited as a solution suitable for adversarial environments. However, it has not been standard practice to follow this advice (mainly due to the fear of unacceptable performance overheads in critical

5. The precise complexity is a function of the load in the table.
6. Due to the Open Source paradigm.

regions of code). Moreover, Smith et al. [9] claim that it may be possible to force the worst case scenario even in the case of universal hashing, by statistically identifying whether two items collide or not, using the detectable processing latency. Another countermeasure is to randomize the Hash table. However, Bar-Yosef and Wool [26] show that there is an attack that can overcome this solution.

4.1 Model

Consider a Hash system consisting of M buckets and N elements, where the Hash function is known to the attackers. In this model the resource-budget of the users is measured by the number of Hash operations (k) they can perform, where Hash operations can be either Insert, Delete or Search. In our analysis we will use three metrics: i) The number of memory references the k operations require during attack time (*In Attack Resource Consumption*), ii) The complexity of performing an arbitrary Hash operation *after the attack has ended*, i.e, after these k Hash-operations are completed (*Post Attack Operation Complexity*). iii) The waiting time of a regular user while performing an arbitrary Hash operation *after the attack has ended*, i.e., after the k Hash-operations are completed (*Post Attack Waiting Time*). Note that both metrics, (i) and (ii), are measured by the number of memory references[7]. In addition, note that the post attack damage endures while the keys inserted by the malicious user are held in the table. For example, in a hash table used as cache the post attack damage lasts until the keys inserted by the malicious user are flushed out. Hence, when evaluating the vulnerability of a designed system, the vulnerability of the system to post attack damage should be considered in the context of the duration in which the post attack damage lasts.

The analysis of the first two metrics is given in this section while the results of the analysis and their practical implications are discussed in Section 7 and depicted in Figure 2. The analysis of the third metric is given in Section 6. In order to carry out the analysis one needs to decide on what type of operations of regular users to base the computation of the denominator of (1), that is $\Delta Perf(R, b)$. In analyzing the Vulnerability Factor, we take the conservative approach and assume that the regular users effect the system by conducting the most time consuming operation types. In both Closed and Open Hash, this is the Insert operation. Thus we receive a lower bound on the actual Vulnerability Factor. Note that as opposed to the traditional elementary operation complexity analysis, in the *Vulnerability Factor* analysis we are also interested in the specific constant and not only in the order of magnitude.

7. It is common practice to count memory references in network algorithms since memory is defacto the bottleneck in most of the systems.

Table of Variables	
M	# of buckets in the table
N	# of existing keys in the table
L	Load in the table ($L = N/M$)
k	# of additional keys inserted by a malicious/regular user (budget)

4.2 The Insert Strategy

Let the **Insert Strategy** (INS) be the malicious users' strategy of performing k insert operations of new elements so that all the inserted elements hash into the same bucket.

Note that in order to create DDoS attacks using the *Insert Strategy*, multiple attackers only need to share one bit of information - the bucket into which they insert the elements.

Theorem 1: In Open Hash and Closed Hash systems, the *Insert Strategy* is an *Optimal Malicious Users Strategy* under the *In Attack Resource Consumption* metric and *Post Attack Operation Complexity* metric.

Theorem 1 is proved separately for Open and Closed Hash in the Appendix.

Remark 1 (The effort of finding the keys): In this work we discuss the case where the selection of elements is done prior to the attack, hence is not included in the attack's budget. If finding the keys can be done only during the attack then the "price" of this effort should be included in the budget and have a realistic proportion (which depends on the system and the capabilities of the attacker) to the "price" of performing an insertion. Note that in Closed Hash, the effort of searching for the right elements can be significantly reduced if the attacker aims at a *range* of buckets instead of a single bucket. For example, it will be two times faster to find 100 elements which hash into a range of two adjacent buckets than into a specific single bucket. Such bucket-range approach can be effective since the reduction in the damage (caused to the system) when aiming at two buckets instead of one is negligible compared to the search effort saved.

4.3 Open Hash

In this subsection we analyze the vulnerability of the Open Hash table (denoted by OH). We define the load factor[8] L of the Open Hash to be equal to $\frac{N}{M}$, which is the average number of elements in a bucket. We assume each bucket is maintained as a linked list. The Insert operation of a new element, which is the most time consuming operation in Hash, requires an expected number of memory references of $L + 1$ [24].

8. This is the load factor as defined in the literature for data structures [24]; it is different from the traffic load.

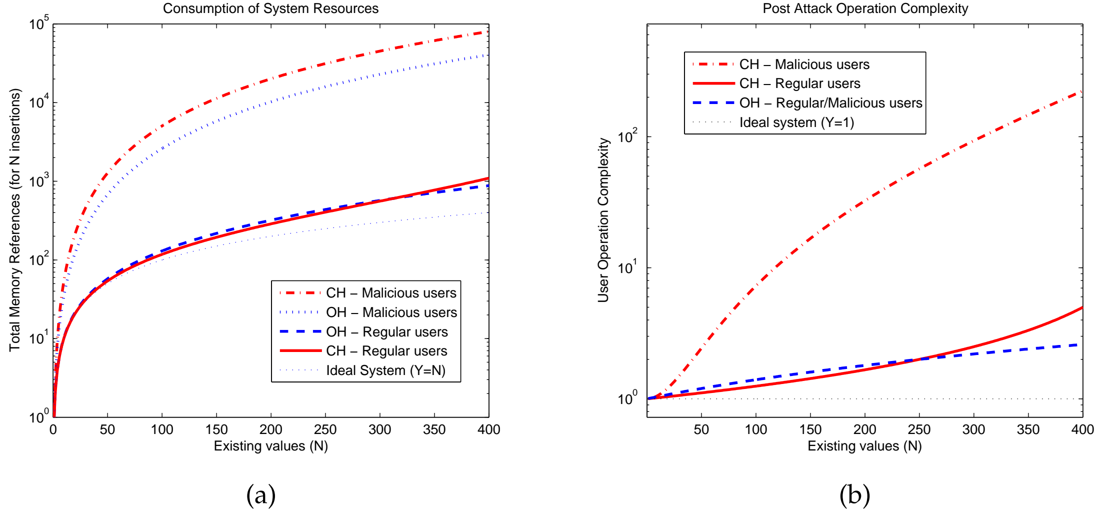


Fig. 2. The performance degradation due to k operations by regular versus malicious users, in Open and Closed Hash under the metrics: (a) *In Attack Resource Consumption* (b) *Post Attack Operation Complexity* . The curves are log scaled and as a function of the number of existing elements N, where $k = N$. The "Ideal System" curves depict the results in an hypothetic ideal invulnerable system where every operation requires exactly one memory access.

4.3.1 In Attack Resource Consumption Metric

The use of the *Insert Strategy* by malicious users on an Open Hash system creates a growing list of elements in one bucket. Every inserted key (of the k keys inserted by the malicious user) has to be compared against all the keys that already exist in the target bucket. Hence, the i-th key inserted has to be compared against the $i-1$ keys previously inserted by the malicious user (and maybe against other already existing keys). Therefore, every insertion by the malicious user suffers an average $O(k)$ complexity (instead of the average case of $O(1)$) thus leading to high in-attack resource consumption.

Theorem 2: Under the *In Attack Resource Consumption* metric the *Vulnerability Factor* of Open Hash is

$$V_{OH}(k) = \frac{k + 2L + 1}{\frac{k+1}{M} + 2L + 2}. \tag{3}$$

Proof: First we prove that $\Delta Perf_{OH}(R, k) = k(1 + L + \frac{k+1}{2M})$. After $i-1$ regular insertions by regular users the load of the table is $L + \frac{i-1}{M}$, therefore the i-th insertion is expected to require $1 + L + \frac{i-1}{M}$ memory references, resulting in total of $k(1+L+\frac{k+1}{2M})$ memory references over k insertions. Next we prove that $\Delta Perf_{OH}(M_{INS}, k) = kL + \frac{k(k+1)}{2}$. The expected number of memory references required by the i-th insertion of malicious users is $L + i$, summing up to $kL+\frac{k(k+1)}{2}$ memory references for the entire insertions sequence. □

Remark 2: Simplifying (3), we may deduce that the parameter k, the magnitude of the attack, is the most dominant parameter that influences the Vulnerability.

Remark 3 (Simplifying population assumption): In order to simplify the already complex analysis, we assume that the existing elements are uniformly distributed in the table. In a real life scenario the elements tend to gather in clusters (due to the nature of linear probing) which increases the average insertion complexity. Using simulations, we confirmed that the impact of this assumption on the final results is insignificant. We measured various performance measures - complexity of a single element insertion, complexity of k insertions by regular/malicious users and the vulnerability of the *In Attack resource Consumption* metric. Each was measured in two scenarios: Scenario

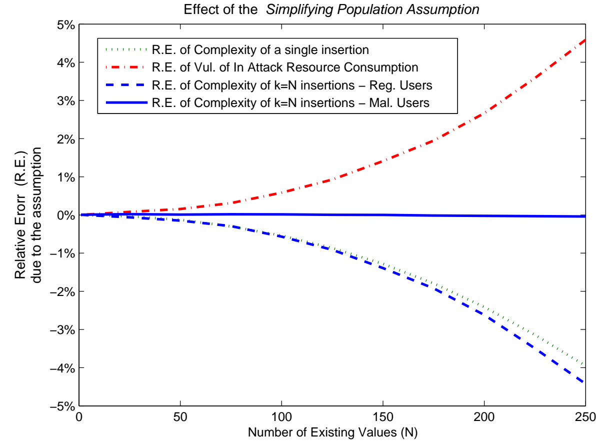


Fig. 3. The effect of using the simplifying population assumption on various performance measures. The plot depicts the relative error of using the simplifying assumption as a function of the number of existing values. This graph is derived from simulation results of a Closed Hash table with 1000 buckets.

A – the existing elements are uniformly distributed in the table; Scenario B – the elements are "naturally" distributed in the table (namely, they were placed according to hashing with linear probing). For each of the performance measures, Figure 3 depicts the deviation of the results in Scenario A from the results in Scenario B. We can see that the relative errors of the complexity of both a single insertion and of k regular insertions are very close and do not exceed -4.5% while the relative error of the complexity of k malicious insertions is almost zero[9] (-0.02%). That is, the negative deviation of $\Delta Perf(R,b)$ is larger than that of $\Delta Perf(M_{st},b)$ when the assumption is used. Therefore, according to Eq. 1 (Section 3), the relative error of the vulnerability is positive and does not exceed $+4.5\%$ (as depicted in Figure 3).

4.3.2 Post Attack Operation Complexity Metric

Theorem 3: Under the *Post Attack Operation Complexity* metric, $V_{OH}(k) = 1$.

Proof: The proof follows from the fact that the expected length of an arbitrarily selected linked list in the Hash table, after the insertion of the k elements, is always $L + k/M$. This holds regardless of their distribution in the buckets (and therefore regardless of the malicious strategy used). □

Remark 4: This characteristic of Open Hash, whereby the expected damage caused by malicious users equals that of regular users, is an important advantage in terms of vulnerability. As Theorem 3 shows, this advantage is well expressed by the Vulnerability Factor.

4.4 Closed Hash

In this subsection we analyze the vulnerability of Closed Hash (denoted by CH). Recall that $L = \frac{N}{M}$ is the fraction of occupied buckets. In practice, L is kept low[10] to keep the Hash table efficient (by decreasing the expected number of rehashing operations needed during a user's request).

For the analysis we assume that collision resolution is done via linear probing; nonetheless, we will explain later that the result derived for the *In Attack Resource Consumption* metric also holds for more sophisticated collision resolution schemes, such as the commonly used double hashing probing. Recalling from Remark 3 that we assume that the existing elements are distributed uniformly over the buckets.

As in Open Hash, in Closed Hash the most time consuming operation is the insertion of a new element. Each such insertion requires on average approximately $\frac{1}{1-L}$ elementary operations where L is the current load in the Hash table (see [24]).

9. The distribution of the elements in the table has a small weight in the operation complexity of a malicious attack.

10. By increasing the number of buckets when the number of elements increases.

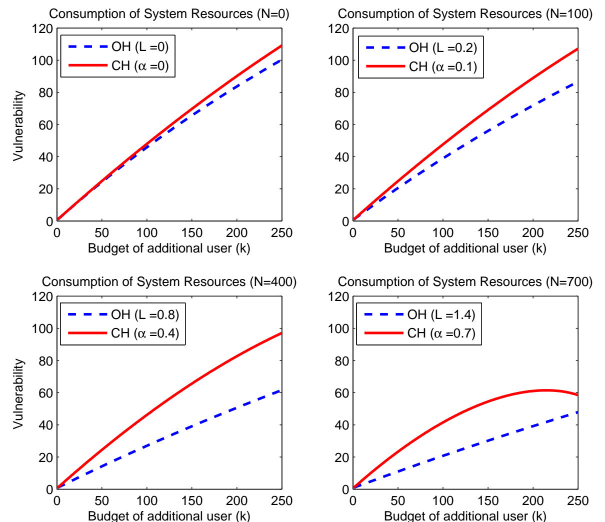


Fig. 4. The vulnerability of the *In Attack Resource Consumption* for different values of N. An ideal invulnerable system has a $V = 1$ curve.

The Vulnerability Factor under both metrics in Closed Hash is influenced by the clustering effect (which also complicates the analysis). In Closed Hash long runs of occupied buckets build up [24]. For example, consider the case of an linear probing Hash function, where the buckets B_1, B_2, B_4 and B_5 are occupied, and an insert operation is made to an element that is hashed to bucket B_1 (where $B_1, B_2, ..., B_M$ are the buckets of the table). The result will be that the element will be inserted into B_3, thus the two clusters $\{B_1, B_2\}$ and $\{B_4, B_5\}$ will be combined into one cluster consisting of 5 elements. These clusters of occupied buckets increase the average insertion time.

4.4.1 In Attack Resource Consumption Metric

The sophisticated attacker's optimal strategy is the *Insert Strategy*, i.e., insert different keys that hash into the same bucket and thus create a long cluster of occupied buckets. The attack creates a cluster which contains the k inserted elements plus the union of multiple clusters that existed before the attack as demonstrated in Figure 5.

Note that under this metric we can avoid making the assumption that a linear probing technique is used. Formally, the attackers search for a pool of keys, such that for every two keys k_1 and k_2 in the pool $h(k_1,i) = h(k_2,i)$ for all i where $h(k,i)$ is the Closed Hash function. A commonly used open addressing Hash function is the double hashing: $h(k,i) = h(k) + ih'(k)$. It is easy to see that even with this double hashing function the attackers can easily find[11] such k_1 and k_2 since they know h and h'.

11. Given k_j, the expected amount of off-line simple hash calculations the attacker needs to conduct to find k_{j+1} is $M^{(2)}$.

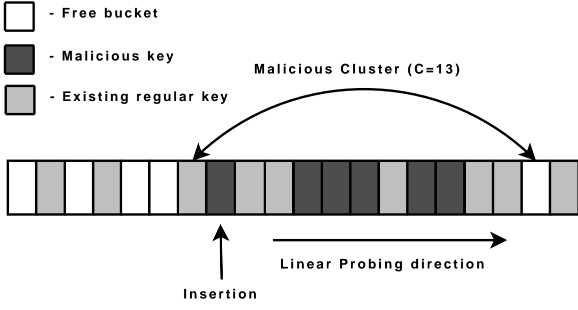


Fig. 5. A toy example of a cluster created by a malicious attack ($M = 20$, $N = 6$, $k = 3$). The “Malicious insertions” arrow points to the bucket where all the malicious keys are initially hashed into.

In Theorem 4, we compute the expected resource consumption and the vulnerability when applying the *Insert Strategy*, but first we derive (the distribution of) the resource consumption of the i-th insertion during a malicious attack in Lemma 1.

Lemma 1: Let X_i be a the number of memory references used to insert the i-th element into the table, then:

$$P(X_i = t) = \frac{\binom{N}{t-i}\binom{M-N}{i}}{\binom{M}{t}} \cdot \frac{i}{t}. \quad (4)$$

Proof: The i-th insertion of the malicious user always traverses the previous $i-1$ elements which were already inserted into the table, but the number of the regular elements it traverses depends on the layout of the elements in the table before the attack takes place. Let $B_1, B_2, ..., B_M$ be the buckets of the table (as defined above), and w.l.o.g let B_1 be the bucket where all the insertions are hashed to when following the *Insert Strategy*. First observe that $X_i = t$ means that $B_1, B_2, ..., B_{t-1}$ were occupied before the i-th insertion while B_t is the free bucket where the inserted element is finally placed. Namely, $X_i = t$ if and only if before the attack has started, there were exactly $t-i$ regular elements in $B_1, ..., B_t$ (which none of them is in B_t). Let Y_t be the number of the regular elements residing in $B_1, ..B_t$, then

$$P(X_i = t) = P(Y_t = t-i) \cdot P(B_t \text{ } is \text{ } empty \text{ } | Y_t = t-i). \quad (5)$$

Y_t follows the hypergeometric distribution with parameters M, N and t and thus, for $i \leq t$: $P(Y_t = t-i) = \frac{\binom{N}{t-i}\binom{M-N}{i}}{\binom{M}{t}}$. In addition, $P(B_t \text{ } is \text{ } empty \text{ } | Y_t = t-i) = \frac{i}{t}$, and according to these equalities and (5) we get (4). □

Theorem 4: Under the *In Attack Resource Consumption* metric the Vulnerability Factor of the Closed Hash is

$$V_{CH}(k) = \frac{\sum_{i=1}^{k} \sum_{t=i}^{i+N} \frac{\binom{N}{t-i}\binom{M-N}{i}}{\binom{M}{t}} \cdot i}{\sum_{i=1}^{k} \frac{1}{1-(L+\frac{i-1}{M})}} \quad (6)$$

Proof: From the complexity of an insert operation of a new element, we get that the resource consumption of k insertions by regular users is $\Delta Perf_{CH}(R, k) = \sum_{i=1}^{k} \frac{1}{1-(L+\frac{i-1}{M})}$. Next we prove the numerator in (6):

$$\Delta Perf_{CH}(M_{INS}, k) = \sum_{i=1}^{k} \sum_{t=i}^{i+N} \frac{\binom{N}{t-i}\binom{M-N}{i}}{\binom{M}{t}} \cdot i. \quad (7)$$

Let X_i be as defined in Lemma 1, then the expected total complexity of the malicious users, $\Delta Perf_{CH}(M_{INS}, k)$, is given by $E(\sum_{i=1}^{k} X_i) = \sum_{i=1}^{k} {X_i}^{(1)}$ where ${X_i}^{(1)} = \sum_{t=i}^{i+N} P(X_i = t) \cdot t$. Using (4) we get (7). □

Remark 5: Simplifying the Vulnerability Factor expression, we obtain the result, similarly to the Open Hash case, that the parameter k is the dominant parameter influencing the Vulnerability.

4.4.2 Post Attack Operation Complexity Metric

In Closed Hash the cluster that was created due to the attack, has a negative effect on the complexity of a user operation even after the attack has ended. Every inserted element which is mapped into a bucket in the cluster (not only to the bucket the attacker used) will require a search that starts at its mapped position and ends at the cluster end[12]. The effect of this on the performance of a user operation will be quite significant. When a key is hashed into a cluster of size C, the expected number of buckets to be probed during the search is approximately $C/2$. In a table with M buckets, the probability to hit this cluster is C/M. That is, the larger the cluster is, the longer the search and, in addition, the higher the probability to hit the cluster in the first place. Therefore, the average search time is approximately proportional the sum of the squares of the cluster sizes in the table ($\frac{1}{2M} \sum_i C_i^2$). This property highly resembles the waiting time in a single-server FIFO queuing system, which is proportional to the ”residual life” of the customer service time, namely proportional to the second moment of the service times. Therefore, unlike the case of the Open Hash, the Vulnerability Factor of the Closed Hash will be (significantly) greater than 1 under this metric, as we will rigorously show in Theorem 5.

First, in Lemma 2 we calculate the complexity of an insertion after a malicious attack has ended. Let

12. This high vulnerability property of Closed Hash may seem to be unique only to a linear probing Closed Hash model due to its primary clustering problem, but it is very hard to implement a simple Hash function which does not cause a clustering problem of some degree.

us look at a Closed Hash table consisting of M buckets, $k \geq 1$ elements inserted by an attacker (using the Insert Strategy) and N elements existing in the table before the attack. Let the random variable Q_M, $1 \leq Q_M \leq N+k+1$, denote the complexity of one regular insertion into the table after the malicious insertions. In Section 6 we use the second moment of Q_M, so in the following lemma we present a general expression where d is the order of the moment.

Lemma 2: Let Q_M, as defined above, be the complexity of an insertion after a malicious attack has ended, then:

$$E[(Q_M)^d] = \sum_{c=k+1}^{N+k+1} \frac{\binom{N}{c-k-1}\binom{M-N}{k+2}}{\binom{M}{c+1}} \frac{k+2}{c+1}[\frac{c}{M}\sum_{s=1}^{c}\frac{1}{c}s^d - (1-\frac{c}{M}) \cdot \sum_{s=1}^{N-k-(c-1)} (\prod_{i=1}^{s-1}\frac{N+K-c-i+2}{M-c-i+1})\frac{M-N-K-1}{M-c-s+1}s^d]. \quad (8)$$

Proof: All the malicious elements concentrate in one large chain of occupied buckets which may contain some or all of the exiting elements. If an element is hashed into this chain, it will eventually be rehashed into one of the two empty buckets trimming the chain. Define the malicious cluster as the chain of consecutive occupied buckets[13] (consisting of the malicious insertions) and the empty bucket that follows them (in the linear probing direction)as seem in Figure 5. Let r.v $k+1 \leq C \leq N+k+1$ be the length of the malicious cluster (the number of elements in the chain is $C-1$). Given that $C=c$, an insertion can be hashed into either the cluster area ($AreaA$) with probability $\frac{c}{M}$ or outside the cluster ($AreaB$) with probability $1-\frac{c}{M}$. Since there is an empty bucket at the end of both areas, it is impossible for an element to be hashed into one area and then rehashed into the other, therefore we are looking for an expression of the following form:

$$E[(Q_M)^d] = \sum_{c=k+1}^{N+k+1} P(C=c)[\frac{c}{M}\sum_{s=1}^{c} P(Q_M=s|C=c, AreaA)s^d + (1-\frac{c}{M})\sum_{s=1}^{N+k-c+2} P(Q_M=s|C=c, AreaB)s^d]. \quad (9)$$

The value of Q_M for an insertion into $AreaA$ depends on the distance between the first bucket where the element was hashed and the empty bucket at the end of the cluster, therefore

$$P(Q_M=s|AreaA, C=c) = \frac{1}{c}, (1 \leq s \leq c). \quad (10)$$

An insertion into $AreaB$ is equivalent to an insertion into a regular Closed Hash table with the same number of buckets and elements as in $AreaB$, therefore according to [24]:

$$P(Q_M=s|AreaB, C=c) = \frac{M-N-k-1}{M-c-(s-1)} \cdot (\prod_{i=1}^{s-1}\frac{N+k-(c-1)-(i-1)}{M-c-(i-1)}). \quad (11)$$

(note that $P(Q_M=1|AreaB) = \frac{M-N-k-1}{M-c}$)

The length of the chain created by k (malicious) insertions plus both of the free buckets trimming it equals to X_{k+2} where X_i is the complexity of the i-th malicious insertion as defined in Section 4.4.1 (This will not be proved here). According to that and the definition of C we get that $C \sim X_{k+2}-1$. Therefore, the probability function of the cluster size is given by:

$$P(C=c) = P(X_{k+2}=c+1) = \frac{\binom{N}{c-k-1}\binom{M-N}{k+2}}{\binom{M}{c+1}}\frac{k+2}{c+1}. \quad (12)$$

Combining (9), (10), (11) and (12) results in (8). □

Theorem 5: Under the *Post Attack Operation Complexity* metric the Vulnerability Factor of Closed Hash is:

$$\Delta V_{CH}(k) = \left(E[Q_M] - \frac{1}{1-N/M}\right) / \left(\frac{1}{1-(k+N)/M} - \frac{1}{1-N/M}\right), \quad (13)$$

where $E[Q_M]$ is as given in (8).

Proof: As already mentioned, the expected complexity of an insertion to a normal Closed Hash table consisting of R elements is given by $\frac{1}{1-R/M}$. Therefore, the expected complexity of insertion when there are only the (N) existing elements is $\frac{1}{1-N/M}$, and after adding k regular insertions it equals $\frac{1}{1-(N+k)/M}$. The expected insertion complexity after k malicious insertions ($E[Q_M]$) is extracted from Lemma 2 using $d=1$ and the proof is completed.

In our case ($d=1$), the value of $\sum_{s=1}^{N+k-c+2} P(Q_m = s|AreaB)s^d$ in the proof of Lemma 2 can be replaced [24] by $\frac{1}{1-\frac{N+k-c+1}{M-c}}$ in order to produce a clearer expression, as we used along the text[14]. By doing so, we get the above expression of $E[Q_M]$. □

5 Sophisticated Attacks on Queueing Mechanisms

Queues are major mechanisms used in a variety of computer and communications systems. Their major objective is to control the system operation when it is under a heavy *traffic load* in order to provide good and fair performance to the various users. For this reason it is natural that they become the target of attacks and their vulnerability should be examined.

To demonstrate the vulnerability of queueing mechanisms to attacks, if not designed properly, we consider a simple case study consisting of a single server

13. Note that it is possible for a chain to start before the bucket which was targeted by the attack.

14. The related plots were drawn according to the exact expression.

(with a single queue) system that operates under the First-Come-First-Served (FCFS) scheduling. This analysis of FCFS is also an important step for us in order to evaluate the vulnerability of practical applications that use Hash (combined with a queue) in the Internet (see Section 6). Consider the M/G/1 queue model with arrival rate λ and service times distributed as a random variable X with first two moments $x^{(1)}$ and $x^{(2)}$ and where the system utilization is $\rho = \lambda x^{(1)}$. Let $W^{(1)}_{scenario}$ be the expected waiting time of an arbitrary arrival where *scenario* can be either of C, R or M, denoting respectively the cases of i) The queue is subject only to normal load by regular arrivals ("clean"), ii) The queue is subject to additional load due to additional regular arrivals, or iii) The queue is subject to additional load due to malicious arrivals. The performance of the system can be evaluated via the mean waiting time experienced by the jobs in equilibrium, which is given by $W^{(1)}_C = \lambda x^{(2)}/(2(1-\rho))$.

One way to attack a FCFS queue is to send large jobs. Consider attackers who behave like regular users who pose additional arrival rate λ_1 and whose job size is also distributed like X. Adding these users to the system changes the mean waiting time to $W^{(1)}_R = (\lambda+\lambda_1)x^{(2)}/(2(1-\rho-\lambda_1 x^{(1)})$, and we have $\Delta Perf_{FCFS}(R, \lambda_1 x^{(1)}) = \frac{(\lambda+\lambda_1)x^{(2)}}{2(1-\rho-\lambda_1 x^{(1)})} - \frac{\lambda x^{(2)}}{2(1-\rho)}$. Now Consider attackers A who send jobs of size $x = Kx^{(1)}$ at rate $\lambda_2 = \lambda_1/K$, that is the additional traffic load they pose is identical to the additional load of the regular users, and thus the attack cost from this perspective is the same (identical budget). The delay in the system, nonetheless, now becomes $W^{(1)}_M = (\lambda x^{(2)} + \lambda_1 K(x^{(1)})^2)/(2(1-\rho-\lambda_1 x^{(1)})$. Thus, we have $\Delta Perf_{FCFS}(M, b = \lambda_1 x^{(1)}) = \frac{\lambda x^{(2)}+\lambda_1 K(x^{(1)})^2}{2(1-\rho-\lambda_1 x^{(1)})} - \frac{\lambda x^{(2)}}{2(1-\rho)}$. Thus if K is chosen to be large enough, $(K(x^{(1)})^2 >> x^{(2)})$ the Vulnerability Factor $V_{FCFS}(b = \lambda_1 x^{(1)}) = \frac{\Delta Perf_{FCFS}(M, b=\lambda_1 x^{(1)})}{\Delta Perf_{FCFS}(R, b=\lambda_1 x^{(1)})}$ can be made as large as one wants, if the system allows arbitrarily large jobs. Thus, if job sizes are not limited, the simple FCFS mechanism is highly vulnerable. We therefore may conclude that queueing mechanisms can be quite sensitive to attacks, if not designed properly. The Vulnerability Factor allows a system designer to evaluate the relationship between the maximal job size allowed in the system and the vulnerability of the system to malicious attack and choose the desired balance between the two.

6 Combining Hash With Queueing

In practical network systems, one would be interested in the combined effect of a Hash table and a queue. This is the case where the requests to the Hash table get queued up in a queue and then be served by the Hash table.

To this end, the queueing analysis given above reveals that the mean waiting time of a request is affected not only by the mean processing time of the other requests but also by their *second moment*. Under the *Post Attack Waiting Time* metric one should revisit the analysis of Open Hash using the *Post Attack Operation Complexity* metric (Theorem 3). Under that analysis the Vulnerability Factor of Open Hash was determined to be 1 since the *mean* length of the linked lists depends only on the load of the table and not on the distribution of the elements in the buckets. Nonetheless, since in the *Insert Strategy*, the malicious users force all items into the same list they can *drastically* increase the *second moment* of the length of an arbitrary list. As a result, we show in this section that they increase the waiting time to pursue a regular user operation even in Open Hash.

Lemma 3: In Open Hash and Closed Hash systems, the *Insert Strategy* is an *Optimal Malicious Users Strategy* under the *Post Attack Waiting Time* metric.

Sketch of Proof: In the *Post Attack Waiting Time* metric, the attacker aims at increasing the mean waiting time of a regular user operation after the attack has ended. This is achieved by maximizing the variance of regular user operations: by inserting all the elements into the same bucket (i.e. the *Insert Strategy*) thus creating in Open Hash one large linked list, and in Closed Hash one large cluster. The detailed proof is carried out using induction on the k malicious user operations.

Theorem 6: Under the *Post Attack Waiting Time* metric

$$V_{OH}(k) \geq 1 + \frac{(1-\lambda L)(k-1)(1-\frac{1}{M})}{(1-\lambda L)(k-1)(\frac{1}{M}) + 2L + 1 - \lambda L(L+\frac{1}{M})}. \tag{14}$$

Proof: Assume an Open Hash table of size M with load factor L. Let $W^{(1)}_{state}$ be the expected waiting time of a table operation according to the *state* of the Hash table. Where *state* can be either of C, R or M, denoting respectively the cases of i) no insertions were made by additional users ("clean"), ii) k elements were inserted by regular users, or iii) k elements were inserted by malicious users.

In addition, assume an M/G/1 queueing model where one processor handles insertion requests to an Open Hash table at rate λ. The Vulnerability is given as:

$$V_{OH}(k) \geq \frac{\Delta Perf_{OH}(M_{INS}, k)}{\Delta Perf_{OH}(R, k)} = \frac{{W_M}^{(1)} - {W_C}^{(1)}}{{W_R}^{(1)} - {W_C}^{(1)}}. \tag{15}$$

Let Y_{state} be a random variable denoting the number of elements in an arbitrary Hash bucket. We will use Y_C, Y_R and Y_M to derive W_C, W_R and W_M, respectively. We assume that the service time for each operation equals the number of the elements in a bucket related to the operation, so according to the M/G/1 waiting time formula we get: ${W_{state}}^{(1)} = \frac{\lambda {Y_{state}}^{(2)}}{2(1-\lambda {Y_{state}}^{(1)})}$, which together with (15) yields:

$$V_{OH}(k) \geq 1 + (1-\lambda(Q_C)^1)((1-\lambda {Y_C}^{(1)})({Y_M}^{(2)} - {Y_R}^{(2)})) \\ /[(1-\lambda Y_C^{(1)})({Y_R}^{(2)} - {Y_C}^{(2)}) + \lambda({Y_R}^{(1)} - {Y_C}^{(1)}){Y_C}^{(2)}] \tag{16}$$

For the individual models we get ${Y_C}^{(1)} = L$, ${Y_C}^{(2)} = L^2 + \frac{M-1}{M}L$, ${Y_R}^{(1)} - {Y_C}^{(1)} = \frac{k}{M}$, ${Y_R}^{(2)} - {Y_C}^{(2)} = \frac{k}{M}(\frac{k-1}{M} + 2L + 1)$, ${Y_M}^{(2)} - {Y_R}^{(2)} = \frac{k}{M}(k-1)(1 - \frac{1}{M})$, which now lead to (14). □

Remark 6: Simplifying (14) we may deduce that the Vulnerability Factor is roughly proportional to k, namely attackers can be very effective. The term $(1 - \lambda L)$ should be considered as a finite non-small number since in common system λL is kept, for stability purposes, below 0.8.

Lemma 4: Let the random variable $Q_{reg(r)}$, $1 \leq Q_{reg(r)} \leq r + 1$, denote the complexity of one regular element insertion into a table of size M with r elements. Then its d-th moment is given by:

$$Q_{reg(r)}{}^{(d)} = \sum_{s=1}^{r+1} \left(\prod_{i=1}^{s-1} \frac{r-(i-1)}{M-(i-1)}\right) \cdot \frac{M-r}{M-r-(s-1)} s^d. \quad (17)$$

Proof: The expression results directly from $P(Q_{reg(r)} = s) = (\prod_{i=1}^{s-1} \frac{r-(i-1)}{M-(i-1)}) \cdot \frac{M-r}{M-r-(s-1)}$ which is proved in [24]. □

Theorem 7: Under the *Post Attack Waiting Time* metric

$$\begin{aligned} V_{CH}(k) \geq 1 + (1 - \lambda(Q_C)^1)((Q_M)^2 - (Q_R)^2)/[(1 - \lambda(Q_C)^1) \\ \cdot({Q_R}^{(2)} - (Q_C)^2) + \lambda((Q_R)^1 - (Q_C)^1)(Q_C)^2], \quad (18) \end{aligned}$$

where ${Q_M}^{(d)}$ follows (8), ${Q_C}^{(d)} = {Q_{reg(N)}}^{(d)}$ and ${Q_R}^{(d)} = {Q_{reg(N+k)}}^{(d)}$ as given in (17).

Proof: Let the random variables Q_C, Q_R, and Q_M denote the complexity of one insertion to a closed hash table in the different states (similar to the definition of Y_{state} in the proof of Theorem 6). The queueing model has not changed, and in the same way (16) was proved for Y_{state} we get (18) for Q_{state}. □

These results for Open Hash emphasize a general observation that can be made regarding any combination a data structure with $O(1)$ amortized complexity and a queue. Although a data structure with $O(1)$ amortized complexity is immune to complexity attacks, when it is combined with a queue it is possible to hurt the system by increasing the variance of the request processing time by the data structure. This leads, as we showed, to an increased waiting time in the queue.

7 Open and Closed Hash: Vulnerability Comparison

We use the results derived above to compare the vulnerability of the Open and Closed Hash models. To conduct the comparison we will adopt the common approach that an Open Hash table with M buckets is performance-wise equivalent to a Closed Hash table with $2M$ buckets[15]. Unless otherwise stated, we use the parameters $M = 500$ and $k = N$, meaning that the additional user always doubles the number of the existing values ($k = N$) in the table.

Figures 6(a) and 6(b) and 6(c) depict the vulnerability of Open and Closed Hash as a function of the number of existing elements (N) under the different metrics. The figures demonstrate a striking and very significant difference between Open and Closed Hash: Closed Hash is much more vulnerable to attacks than Open Hash. Regarding the resource consumption during an attack, we can see in Figure 6(a) that Closed Hash is more vulnerable than Open Hash, but both are vulnerable. Nevertheless, Figure 6(b) depicts that Open Hash is invulnerable ($V = 1$) for the post attack operation complexity while Closed Hash is vulnerable due to the clustering created by the attack. The performance of the open and closed Hash for the regular users *waiting time* metric is provided in Figure 6(c). The figure reveals two interesting and important properties of the system. First, under this metric Open Hash is much more vulnerable than under the *Post Attack Operation Complexity* metric. This is due to the fact that the delay is proportional to the *second moment* of chain length (and not to the first moment as in Figure 6(b)). Second, Closed Hash is drastically more vulnerable (the y-axis scale is logarithmic, reaching values of 10^4, the queue is no longer stable for $N > 237$) resulting from the compounded effects of the tendency of requests to address large clusters and of the mean queueing delay to depend on the *second moment* of the hash operation times. Hence this performance degradation may reach the level of a total denial of service.

In Figure 4 we study in more detail the vulnerability of Hash using the *In Attack Resource Consumption* metric, as a function of the attack size (k) and the number of existing elements in the table (N), which defines the load before the attack (while in all other figures we used $N = k$). The figure demonstrates that when the table is lightly populated (upper figures) the Vulnerability Factors of both Open and Closed are quite similar; the similarity stems from the fact that in Closed Hash, with high likelihood, a simple chain, consisting only of the k elements of the attack, will be formed (and thus will be similar in performance to Open Hash). The more common/realistic cases are depicted in the lower figures, representing medium to high Hash population (load). In these cases the vulnerability of Closed Hash becomes much larger due to the clustering effect[16].

In order to show that the Vulnerability Factor expresses well the difference in the performance of the

15. It is common to consider them equivalent with respect to the space they require due to the pointers required by Open Hash.

16. In Figure 4, one might question why at high load ($N = 700$, $k = 250$) the vulnerability metric of Closed Hash starts decreasing and approaches that of the Open-Hash. The reason is that at this load, the Closed Hash table is almost full and probably contains big clusters, thus the insertion complexity for a regular user becomes close to that of a malicious user, causing the Vulnerability to decrease.

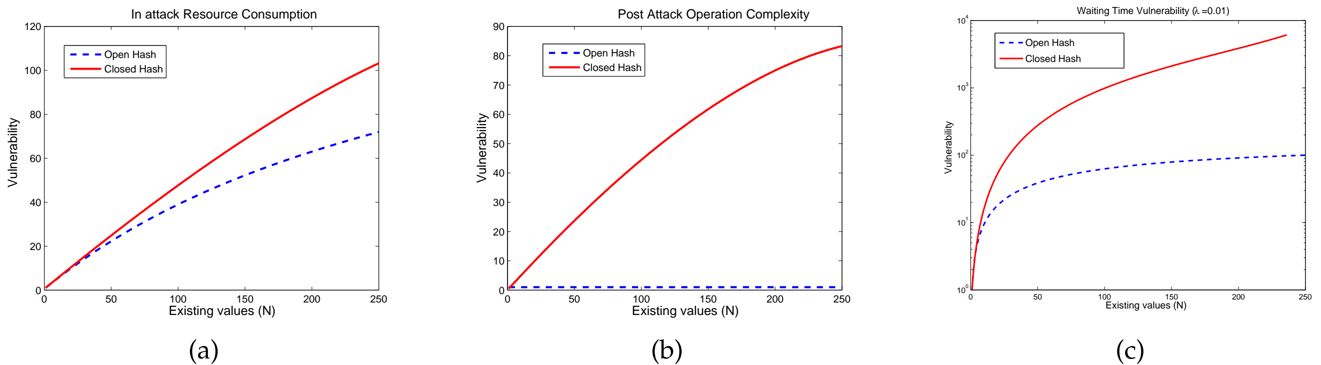


Fig. 6. Vulnerability comparison of Open and Closed Hash as a function of N, the number of existing elements and where k equals N in Metric (a) *In Attack Resource Consumption* (b) *Post Attack Operation Complexity* (c) *Post Attack Waiting Time* . An ideal invulnerable system has a $V = 1$ curve (as the curve for Open Hash in (b)).

system under attack, we show in Figures 2(a) and 2(b) the performance degradation due to k operations queried by malicious users and by regular ones in the first two metrics. In Figure 2(a) we show that in the *In Attack Resource Consumption* metric, for regular users up to the point of $N = 313$, the Closed Hash system handles the k additional elements better and only after this point Open Hash becomes superior, the performance of both is not far from that of the "Ideal System" where every insertion requires exactly one memory access. Nonetheless, when the additional users are malicious the Open Hash system is, by far, more efficient for the whole range of parameters. In Figure 2(b) we show that in the *Post Attack Operation Complexity* metric the difference between the performance of the two types of Hash is even more significant. When the additional users are malicious, the Open Hash system is by far more efficient for the whole range of the parameters and stays. Note that in both figures the vertical axis is depicted in log scale, namely, the gap between the consumption of regular and malicious users is very large in both models.

Practical Implications: The analysis above demonstrates the importance of the vulnerability metric since it exposes major performance gaps between Open and Closed Hash. This is in contrast to the traditional performance measures that cannot identify these gaps and thus identify the two systems as performance-wise equivalent. Another implication is that 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 one would double the size of the Closed Hash table (rehashing) when the table reaches $70\%-80\%$ load. However, in a hostile environment the rehashing must be done much earlier. For a specific Hash state we can calculate the maximum magnitude of an attack so that the queue remains stable after the attack has ended. Above this stability point, the status of the Hash table is such that the arrival rate of regular user operations, times their expected processing time, is greater than one. Therefore the system cannot cope with the users requests, and hence the users suffer from a total denial of service and not just partial performance degradation. For example consider Figure 6(c) where the state of the Hash table is $M = 1000$ and $k = N$. In this system the stability point is $N = 237$ (the plot ends at this point). Namely, with less than 48% $(N + k = 474)$ of buckets occupied, the queue is no longer stable. The attackers can destabilize the queueing system even if the expected cluster size is not too large, since the waiting delay is proportional to the *second moment* of the cluster size.

8 Conclusion and Future work

The performance of today's networks and computers is highly affected by DDoS attacks launched by malicious users. This work originated in the recognition that in order to properly evaluate the performance and qualities of these systems one must bridge between the world of security and that of quality of service, and add to the traditional performance measures a metric that accounts for the resilience of the system to malicious users. We proposed such a Vulnerability Factor that measures the relative effect of a malicious user on the system. Using the Vulnerability measure we made interesting observations regarding the vulnerability of Hash systems: 1) The Closed Hash system is much more vulnerable to DDoS attacks than the Open Hash system. 2) After an attack has ended, regular users still suffer from performance degradation. 3) This performance degradation may reach the level of a total denial of service; We can calculate the exact magnitude of attack that can cause it. 4) An application which uses a Hash table in the Internet, where there is a queue in front of the Hash Table, is highly vulnerable. In this case the waiting time the regular users suffer is proportional to the second moment, and hence the attack size has a duplicate effect on the Vulnerability. This study has only just

begun and we believe that much more work is needed in order to evaluate many other traditional systems and examine their vulnerability.

REFERENCES

[1] C. Labovitz, D. McPherson, and F. Jahanian, "Infrastructure Attack Detection and Mitigation," in *Proceedings of ACM SIGCOMM Conference on Applications*, Aug. 2005.

[2] *TCP SYN Flooding and IP Spoofing Attacks*, CERT, Sep. 1996, http://www.cert.org/advisories/CA-1996-21.html.

[3] S. A. Crosby and D. S. Wallach, "Denial of Service via Algorithmic Complexity Attacks," in *Proceedings of USENIX Security Symposium*, Aug. 2003.

[4] A. Bremler-Barr, H. Levy, and N. Halachmi, "Aggressiveness Protective Fair Queueing for Bursty Applications," in *Proceedings of IEEE IWQoS International Workshop on Quality of Service*, Jun. 2006.

[5] M. Guirguis, A. Bestavros, and I. Matta, "Exploiting the Transients of Adaptation for RoQ Attacks on Internet Resources," in *Proceedings of IEEE ICNP International Conference on Network Protocols*, Mar. 2004.

[6] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang, "Reduction of Quality (RoQ) Attacks on Internet End-Systems," in *Proceedings of IEEE INFOCOM International Conference on Computer Communications*, Mar. 2005.

[7] A. Kuzmanovic and E. W. Knightly, "Low-Rate TCP-Targeted Denial of Service Attacks (The Shrew VS. the Mice and Elephants)," in *Proceedings of ACM SIGCOMM Conference on Applications*, Aug. 2003.

[8] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang, "Reduction of Quality (RoQ) Attacks on Dynamic Load Balancers: Vulnerability Assessment and Design Tradeoffs," in *Proceedings of IEEE INFOCOM International Conference on Computer Communications*, May 2007.

[9] R. Smith, C. Estan, and S. Jha, "Backtracking Algorithmic Complexity Attacks Against a NIDS," in *Proceedings of ACSAC Annual Computer Security Applications Conference*, Dec. 2006.

[10] A. Jaquith, *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. Addison-Wesley Professional, 2007.

[11] S. Lippman and S. Stidham, "Individual versus Social Optimization in Exponential Congestion Systems," *Operations Research*, vol. 25, pp. 233–247, 1997.

[12] U. Ben-Porat, A. Bremler-Barr, and H. Levy, "Evaluating the Vulnerability of Network Mechanisms to Sophisticated DDoS Attacks," in *Proceedings of IEEE INFOCOM International Conference on Computer Communications*, Apr. 2008.

[13] *Distributed Denial of Service Tools*, CERT, Dec. 1999, http://www.cert.org/incident_notes/IN-99-07.html.

[14] M. D. McIlroy, "A Killer Adversary for Quicksort," *Software–Practice and Experience*, pp. 341–344, 1999.

[15] T. Peters, "Algorithmic Complexity Attack on Python," May 2003, http://mail.python.org/pipermail/python-dev/2003-May/035916.html.

[16] M. Fisk and G. Varghese, "Fast Content-Based Packet Handling for Intrusion Detection," University of California at San Diego (UCSD) La Jolla, CA, USA, Tech. Rep., 2001.

[17] F. Weimer, "Algorithmic Complexity Attacks and the Linux Networking Code," http://www.enyo.de/fw/security/notes/linux-dst-cache-dos.html.

[18] T. Moscibroda and O. Mutlu, "Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems," in *Proceedings of USENIX Security Symposium*, Jun. 2007.

[19] C. Castelluccia, E. Mykletun, and G. Tsudik, "Improving Secure Server Performance by Re-balancing SSL/TLS Handshakes," in *Proceedings of USENIX Security Symposium*, Apr. 2005.

[20] J. Bellardo and S. Savage, "Abstract 802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions," in *Proceedings of USENIX Security Symposium*, Jun. 2003.

[21] U. Ben-Porat, A. Bremler-Barr, H. Levy, and B. Plattner, "On the Vulnerability of the Proportional Fairness Scheduler to Retransmission Attacks," in *Proceedings of IEEE INFOCOM International Conference on Computer Communications*, Apr. 2011.

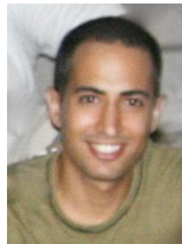
[22] U. Ben-Porat, A. Bremler-Barr, and H. Levy, "On the Exploitation of CDF Based Wireless Scheduling," in *Proceedings of IEEE INFOCOM International Conference on Computer Communications*, Apr. 2009.

[23] C.-F. Yu and V. D. Gligor, "A Formal Specification and Verification Method for the Prevention of Denial of Service," in *IEEE Symposium on Security and Privacy*, May 1998.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2001.

[25] J. L. Carter and M. N. Wegman, "Universal Classes of Hash Functions," *Journal of Computer and System Sciences (JCSS)*, pp. 143–154, 1979.

[26] N. Bar-Yosef and A. Wool, "Remote Algorithmic Complexity Attacks Against Randomized Hash Tables," Master's thesis, Tel-Aviv University, Tel-Aviv, Israel, 2006.



Udi Ben-Porat received his B.Sc (2005) and M.sc (2009) degrees in Computer Science from Tel Aviv University. In 2009 he started his doctoral studies at the Computer Engineering and Network Laboratory (Communication Systems group), ETH Zurich. His interests include Wireless Networks, Communication Systems, Performance and Vulnerability of shared systems.

Dr. Anat Bremler-Barr received the Ph.D degree (with distinction) in computer science, from Tel Aviv University. In 2001 she co-founded Riverhead Networks Inc., a company that provides systems to protect from Denial of Service attacks. She was the chief scientist of the company. The company was acquired by Cisco Systems in 2004. In 2004 she joined the School of Computer Science at the Interdisciplinary Center Herzliya. Her research interests are in computer networks and distributed computing. Her current works focused on designing routers and protocols that support efficient and reliable communication.

Prof. Hanoch Levy received the B.A. degree in Computer Science with distinctions from the Technion, Israel Institute of Technology in 1980 and the M.Sc. and the Ph.D. degrees in Computer Science from University of California at Los Angeles, in 1982 and 1984, respectively. From 1984 to 1987 he was a member of technical staff in the Department of Teletraffic Theory at AT&T Bell Laboratories. From 1987 he has been with the School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel. In 1992 - 1996 he was at the School of Business and RUTCOR, Rutgers University, New-Brunswick, NJ, and in 2007 - 2008 he was at the Communication Systems group, ETH, Zurich, both on leave from Tel-Aviv University. His interests include Computer Communications Networks, Wireless networks, Performance Evaluation of Computer Systems and Queuing Theory.