Validating Damage Assessment: A Simulation-Based Analysis of Blind Write Lineage in Fog Computing

Mariha Siddika Ahmad
Electrical Engineering and Computer Science Department
University of Arkansas
Fayetteville, AR 72701 USA
ma135@uark.edu

Brajendra Panda
Electrical Engineering and Computer Science Department
University of Arkansas
Fayetteville, AR 72701 USA
bpanda@uark.edu

***Abstract*— In order to solve problems like temporal lag, communication overhead, and the requirement for computational and storage resources to be closer to both the ground and end users, the idea of fog computing evolved as an extension of cloud computing. Due to its large number of interconnected nodes, this system is susceptible to cyberattacks. Damage from the attack swiftly spreads to other nodes and their data items when valid transactions cause modifications using the value of a compromised object, thus impairing the system's real-time functions. This research proposes a fast damage assessment approach to provide users access to unaffected nodes while accelerating system recovery. In this paper, we provide various algorithms along with simulated results to efficiently identify and mitigate damage, ensuring the system's resilience and continuity. Our proposed method demonstrates significant improvements in damage assessment speed and efficiency compared to traditional approaches, with simulation results consistently showing substantially fewer data item reads required during the recovery process across various scenarios.**

Keywords- Fog computing security; blind write; data dependency; damage assessment; malicious transaction.

I. Introduction

While cloud computing brought advantages like processing power and communication, it also raised concerns about data security and user experience due to delays. Fog computing was developed as an extension of cloud computing to tackle these issues. It brings the needed resources closer to users but inherits security and privacy risks from traditional cloud systems. In fog environments, the interconnected nature and vast amount of data create a bigger attack surface, allowing damage to spread rapidly. This is especially worrisome for real-time processing, where breaches can have a swift and significant impact. For critical systems like those used by emergency services, hospitals, and police, fast recovery of compromised data is essential. Unfortunately, traditional recovery methods, which involve shutting down systems for analysis and restoration, are not suitable for real-time fog systems.

Traditional logs struggle to track data access during attacks, hindering recovery. Blind write operations are those that update data without reading it. These can be handily used to accelerate recovery. This research builds on prior work on blind write's role in recovery efficiency [1]. We explore the damage assessment process and present the effectiveness of our approach through simulation. We show that by using blind writes, data recovery can be automated and becomes significantly faster, eliminating manual data checks.

This paper is structured as follows: the following section will explore the motivation behind this research. Section 3 will examine relevant past research connected to this endeavor. In Section 4, we will delve into the specifics of the model. Section 5 will provide a broad discussion of damage assessment for blind write lineage along with their corresponding algorithms. Then, in Section 6, the simulated results are shown. Finally, Section 7 will wrap up the paper with the conclusion.

II. Motivation

Distributed fog systems strategically store large amounts of data near users, enabling real-time services in critical areas. Their speed and reliability make them ideal for essential computing infrastructure across various organizations, from local emergency services to large enterprises. However, the sensitive data they store makes them attractive targets, and their inherent vulnerabilities pose significant security challenges.

Fog systems' strength lies in their interconnectedness, but this very feature creates a vulnerability. Like a chain breaking at its weakest link, a single compromised node can trigger a domino effect, crippling interconnected systems. These systems are also heterogeneous, meaning different nodes have varying software and data. Additionally, the sheer volume of data across all connected nodes makes recovery extremely complex after an attack. This complexity leads to significant delays, a major issue considering the real-time nature of the services fog systems provide.

A successful attack on a single fog node can wreak havoc beyond its local database. Corrupted data can spread like a virus as legitimate transactions unknowingly read tainted information and update healthy data based on these bad values. This domino effect rapidly infects other interconnected nodes, creating a snowballing problem. Recovering from such an attack becomes a race against time, as these systems rely on real-time functionality. A swift and accurate recovery mechanism is essential, not just for system survival but also to minimize service disruptions.

The potential impact on critical infrastructure includes:

- **Emergency Services**: A compromised fog node could lead to incorrect dispatch of information, delaying response times and potentially costing lives.

ISBN: 978-1-68558-206-7

- **Healthcare Systems**: Corrupted patient data could result in misdiagnosis or improper treatment plans.
- **Financial Institutions**: Tainted transaction data could cause widespread financial losses and erode customer trust.
- **Smart City Infrastructure**: Compromised traffic management systems could lead to gridlock or increased accident risks.

Traditional recovery methods, which often involve system-wide shutdowns and time-consuming manual checks, are simply not feasible for these real-time, mission-critical systems. The need for a solution that can rapidly assess damage, isolate compromised data, and restore system functionality is paramount.

This paper proposes a novel method for recovering data in fog computing systems that manage vital information. We highlight the urgency of real-time recovery during cyberattacks, leveraging the concept of blind writes to accelerate the damage assessment process. Our approach aims to minimize downtime, reduce the spread of corrupted data, and ensure the continuity of essential services even in the face of sophisticated attacks.

By addressing these critical challenges, our research contributes to building more resilient fog computing infrastructures capable of withstanding and swiftly recovering from cyber threats, ultimately safeguarding the vital services that modern society increasingly depends upon.

III. Related Work

Following the success of cloud computing, fog and edge computing are emerging as the next frontier. Research on fog computing's role in the Internet of Things is well-established by Bonomi et al. in [2] as well as several insightful surveys exploring its key issues and potential by Mouradian et al. in [3] and Vaquero et al. in [4]. Security remains a major concern, as evidenced by various studies by Sun et al. [5] and Mukherjee et al. [6]. In addition, in reference [7], Wu et al. explore security vulnerabilities in critical infrastructure data storage and management systems. Additionally, research by Viganò et al. in [8] highlights vulnerabilities specific to critical infrastructure data management systems.

Beyond general security, researchers like Kotzanikolaou et al. [9] have investigated targeted risk assessment models for cascading failures in critical infrastructure. Others have emphasized how Cyber-Physical Systems introduce new attack vectors in data-rich environments like Ding et al. in [10]. Notably, Rehak et al. [11] propose a valuable model depicting interconnected elements within an infrastructure system. This model's focus on dependencies closely resembles the interconnected nature of fog computing systems, offering insights into potential cascading damage from attacks.

Database attacks can have a ripple effect, corrupting healthy data through seemingly valid transactions unaware of the compromise. Post-attack recovery is crucial, relying heavily on system logs to identify affected data and initiate recovery procedures. The concept of blind writes, updating data without reading them, has been explored by various researchers. Stearns et al. [12] define it as writing data without a prior read request, highlighting the lack of a preliminary check. Mendonca et al. [13] emphasize that during a blind write operation, data copies are modified regardless of their original values. Similarly, Burger et al. [14] focus on the absence of a pre-write read operation inherent to blind writes.

Rapid damage assessment is vital for fog system recovery. Existing methods leverage transaction or data dependencies for this purpose by Ammann et al. in [15] and Tripathy et al. in [16]. Recovery is equally important, as evidenced by research on database recovery after attacks by Panda et al. [17] and efficient damage assessment algorithms by Haraty et al. [18]. Notably, Haraty et al. [18] presents a memory and time-efficient algorithm that effectively handles blind writes, minimizing attack impact and enabling swift, accurate recovery.

The field of damage assessment has seen prior research exploring the potential of blind writes for faster identification of compromised data [1]. This paper takes that concept a step further. We introduce a new model specifically tailored for fog computing systems, a domain where rapid damage assessment is paramount due to their interconnected nature and the potential for swift propagation of attacks. Our proposed model leverages blind writes for efficient damage assessment, and we will further validate its effectiveness through the inclusion of simulated results. By incorporating simulations, we aim to demonstrate the model's ability to quickly pinpoint compromised data after a cyberattack in a fog computing environment.

Fog computing's distributed nature compels us to explore damage assessment techniques from distributed systems research. Existing work in this area offers promise for fog computing. For instance, Alshehri et al. [19] outline a blockchain-based method designed to prevent a malicious fog node from affecting other nodes in the network. The authors propose a model that uses blockchain technology and Ciphertext-Policy Attribute-Based Encryption (CP-ABE) to create fog federations that enable secure and distributed authorization among fog nodes. This approach aims to reduce time delays and communication overhead between fog nodes and cloud servers by allowing fog nodes within the same federation to conduct distributed authorization processes using smart contracts on the blockchain. By adapting these techniques, specifically by integrating blockchain-based authorization mechanisms and encryption methods such as CP-ABE, fog systems can enhance their defense against cyberattacks while maintaining system performance and scalability. This approach enables more secure and efficient damage recovery in fog computing environments, where timely and coordinated responses to threats are crucial for resilience.

Our research introduces the concept of blind write lineage to address security challenges in fog computing. This model efficiently traces data dependencies and rapidly assesses damage by leveraging the characteristics of blind writes. Unlike traditional methods that rely on log analysis, our approach offers a more effective solution for real-time damage assessment in interconnected fog systems, which are

ISBN: 978-1-68558-206-7

particularly vulnerable to cascading failures and rapid damage propagation.

IV. Blind Write Lineage Model

The previous research [1] leverages blind writes for rapid damage assessment in fog computing systems. Blind writes update data without first reading their existing values. A key concept is data dependency, where one data item relies on the value of another for its update.

[1] introduces the concept of blind write lineage, which tracks data items solely dependent on a blindly written item or its descendants. To facilitate damage assessment and recovery, two crucial data structures are maintained:

Blind Write (BW) List: List which includes blindly written data items, timestamps, and transaction numbers.

Blind Write Lineage (BW Lineage) List: Tracks the lineage of data items solely dependent on blindly written items or their descendants. Represented as [Parent_node → Child_node].

Blind write lineage is a method for swift damage assessment in fog computing systems. It capitalizes on blind writes, where data updates occur without first retrieving the existing value. The previous paper [1] explored two primary scenarios within blind write lineage:

Case 1: Single-parent/Single-child Lineage (Simpler Scenario)

This case represents a simpler scenario where data items are updated sequentially, with each item relying on a single predecessor. The lineage of affected data items can be efficiently traced back to the original blindly written item.

Case 2: Multipath Lineage (More Complex Scenario)

This case presents a more intricate scenario where a child node might have multiple parent nodes, and vice versa. Data items can also be updated by leveraging multiple arguments. This complexity necessitates a more refined approach to damage assessment.

In a fog computing system, the integrity of data relies heavily on the relationships between various data items. When an attack occurs, it is crucial to identify how data has been compromised by analyzing transaction logs. These logs store the sequence of operations performed on data items, including read and write operations. In the context of blind writes, where data is updated without reading its prior value, it becomes especially challenging to trace which items are affected by compromised data.

To efficiently perform damage assessment, the system needs to track how data dependencies unfold. Each time a data item is blindly written, its dependent items—those that rely on it for updates—become potential candidates for compromise. By analyzing the transaction logs, the system can trace these dependencies and organize them into subgraphs, which represent different clusters of related data items.

The analysis of transaction logs in a fog computing system can reveal complex data dependencies. These dependencies can be visualized as a graph, where:

- Nodes represent data items
- Edges represent dependencies between data items (i.e., one data item being used to update another)

However, this overall graph is not necessarily fully connected. Instead, it often consists of multiple disconnected subgraphs. Each of these subgraphs (G_1, G_2, G_3, etc.) represents a distinct "blind write lineage" - a chain of data updates originating from a blindly written data item.

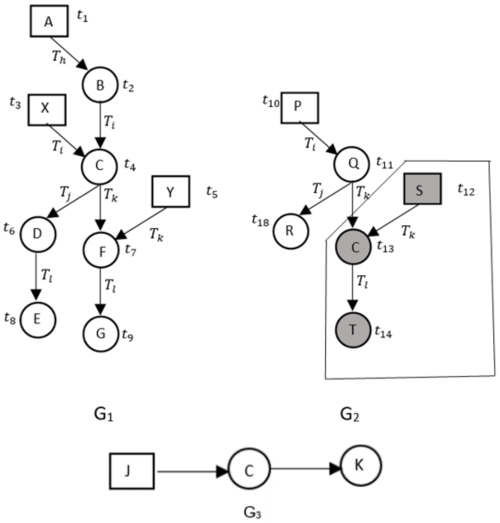

Figure 1. Multiple subgraphs in the data dependency (G) [1].

The analysis of transaction logs can reveal multiple disconnected subgraphs within the overall data dependency graph (Figure 1). These subgraphs, denoted as G_i (where i represents a specific subgraph), collectively form the set G. Each subgraph (G_i) represents a distinct blind write lineage. For Figure 1, G would be: G: {G_1, G_2, G_3}. This structure allows for efficient assessment of damage even in scenarios with multiple attack points.

One algorithm focuses on the simpler scenario of single-parent/single-child lineage. It operates by first checking if the initially compromised data item is present on the Blind Write (BW) list. If found, the algorithm leverages the Blind Write Lineage (BW Lineage) list to identify all subsequent data items affected by the attack. The final output is a comprehensive list of Damaged Data Items that require remediation.

The next algorithm tackles the more general scenario of multipath lineage. It achieves this by first identifying distinct subgraphs within the overall data dependency graph. For each subgraph, the algorithm defines three crucial sets:

Blind Write Set (*BWS_i*): This set encompasses all blindly written data items within the subgraph, along with their corresponding timestamps. These blindly written items serve as the root cause of potential damage.

Children Data Set (*CDS_i*): This set comprises all data items that are dependent on one or more elements within the ***BWS_i*** of the same subgraph. It also includes timestamps for each data item's update. These data items are considered potentially compromised due to their dependency on blindly written elements.

Damaged Set (*D*): These items are considered damaged because they were created by an attacker transaction. The

Copyright (c) IARIA, 2024. ISBN: 978-1-68558-206-7

damaged set is like a family tree, where each data item is a parent node in their respective subgraphs. This is because the assumption is that attacker transactions create data blindly.

By meticulously constructing these sets for each subgraph based on transaction data, the last algorithm lays the groundwork for effective damage assessment. The resulting subgraphs, along with the *BWS_i* and *CDS_i* sets, provide valuable insights into the extent of the attack and the data items that require further investigation or restoration.

V. Damage Assessment

For damage assessment, time (t_a) as in attack time and for every data item last updated time ($t_{last\ updated\ time}$) and graph(G) must be taken into consideration for damage assessment. As was mentioned in the previous paper [1], the same data item can be updated in different transactions at different times. So, time is very crucial here to find the damaged data items and if that damaged item has been used before or after the attack, depending on the time, it can be decided if the damaged item should be recovered or not. Again, the damaged graph is needed as in to differentiate if the same data item is updated at the same time, it would be much easier for assessment. For this purpose, the final updated time for each data item and their corresponding subgraph would be listed in a table (Table II). There would be one table for Graph set G_i (Table I). Suppose for this example let's check the final updated timetable and the BWSi and CDSi set:

TABLE I. Sets

G_i	**G_1**	**G_2**	**G_3**
BWS_i	{(A,$\boldsymbol{t_1}$), (X,$\boldsymbol{t_3}$), (Y,$\boldsymbol{t_5}$)}	{(P,$\boldsymbol{t_{10}}$), (S,$\boldsymbol{t_{12}}$)}	{(J,$\boldsymbol{t_{15}}$)}
CDS_i	{(B,$\boldsymbol{t_2}$), (C,$\boldsymbol{t_4}$), (D,$\boldsymbol{t_6}$), (E,$\boldsymbol{t_8}$), (F,$\boldsymbol{t_7}$), (G,$\boldsymbol{t_9}$)}	{(Q,$\boldsymbol{t_{11}}$),(C,$\boldsymbol{t_{14}}$), (T,$\boldsymbol{t_{15}}$)}, (R,$\boldsymbol{t_{18}}$)}	{(C,$\boldsymbol{t_{16}}$), (K,$\boldsymbol{t_{17}}$)}
D	{(S,$\boldsymbol{t_{12}}$)}		

TABLE II. Final Updated Time Table

Data Items	A	B	X	Y	D	E	F	G
$t_{\text{Last Updated}}$	$\boldsymbol{t_1}$	$\boldsymbol{t_2}$	$\boldsymbol{t_3}$	$\boldsymbol{t_5}$	$\boldsymbol{t_6}$	$\boldsymbol{t_7}$	$\boldsymbol{t_8}$	$\boldsymbol{t_9}$
Graph	G_1	G_1	G_1	G_1	G_1	G_1	G_1	G_1

Data Items	P	Q	S	T	J	***C***	K	R
$t_{\text{Last Updated}}$	$\boldsymbol{t_{10}}$	$\boldsymbol{t_{11}}$	$\boldsymbol{t_{12}}$	$\boldsymbol{t_{14}}$	$\boldsymbol{t_{15}}$	$\boldsymbol{t_{16}}$	$\boldsymbol{t_{17}}$	$\boldsymbol{t_{18}}$
Graph	G_2	G_2	G_2	G_2	G_3	G_3	G_3	G_2

In all three subgraphs (Figure 1), *C* is found to be updated at t_4, t_{13} *and* t_{16}. But in Table II, the final update of *C* is listed which is t_{16} and it appears in subgraph G_3 (shaded part).

It is possible for the same data items to be blindly written by multiple transactions. For instance, let's consider the data item S, which could be blindly written in all the subgraphs (G_1, G_2, and G_3). In such a scenario, all the Blind Write Sets (BWS_i) for these subgraphs would contain "*S*." However, if the update time is not considered within the set, all the subgraphs will be deemed damaged. So, the time of the update of each data item has been included in the BWS_i and CDS_i as an ordered pair. Thus, in this example, when the initial damaged set D, {(S,t_{12})}, is intersected with the Blind Write Sets (BWS_i) of all the subgraphs in the system, only G_2 would be identified as damaged, as the ordered pairs match.

In the case of G_1 and G_2, G_1 has used a non-damaged *C*. Since it is in a different graph it has no connection with subgraph G_2 hence this value is independent of that value of *C* there. It is evident that data item *C* is a child of *S*, the initially maliciously modified data item, implying that *C* is damaged. However, in G_3, *C* has been modified using a blindly written data item, *J*. Given that these subgraphs are isolated and unrelated to each other, it is deduced that *C* in G_3 has already been recovered and can be released for use.

Upon establishing that a specific graph is affected, the time of update for every child of the initial damage is referred to as the affected time. For instance, in G_2, if *S* is the initial damage, then the time of update for *C* is denoted as t_{13}, which represents the affected time for *C*, and for T, the affected time is t_{14}. These affected times can also be found in the CDS_i. Another case to be mindful of is the possibility of a specific data item being recovered within the same damaged graph. Let's illustrate this scenario with an example to provide clarity:

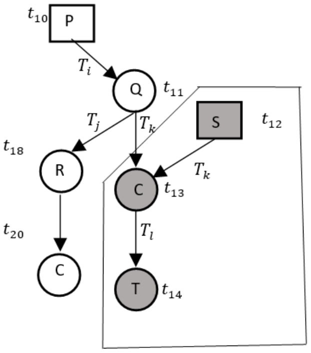

Figure 2. An example showing a data item being damaged and recovered in the same Subgraph G_2.

In Figure 2, it is evident that data item *C* was initially damaged at t_{13}. However, it undergoes modification again at t_{20}, transpiring within the same damaged graph. Notably, this time, *C* has a parent data item *R* that remains undamaged. Consequently, *C* is successfully recovered within the same damaged graph.

In a damage assessment scenario using the following algorithm for case 2, imagine we have a subgraph where the initial damaged data item is *A*, which belongs to the Blind Write Set (BWS) of the subgraph. The system starts by identifying that the subgraph is compromised because *A* intersects with the BWS. Next, it evaluates the Child Data Set (CDS), which contains dependent data items, such as *C* and D*D*. Since *C* depends on *A*, it is flagged for

Courtesy of IARIA Board and IARIA Press. Original source: ThinkMind Digital Library https://www.thinkmind.org

ISBN: 978-1-68558-206-7

potential damage and added to the Potential Damaged List (PDL). Moving further, *D* depends on *C*, and as *C* is already marked for damage, *D* is also added to the PDL. The algorithm then checks the final updated times for *C* and *D* against the attack time and the affected time. If the last updated time of an item is equal to the affected time (e.g., *C*), it is confirmed as damaged and retained for further evaluation. For items like *D*, if its last update is after the affected time, the algorithm checks if its parent (in this case, *C*) is damaged. Since *C* is indeed compromised, *D* is also classified as damaged. Finally, the algorithm outputs a list of damaged data items (*C* and *D*) for further recovery processes. This structured approach efficiently isolates and assesses damage propagation through data dependencies.

Algorithm: (Evaluate Subgraph Damage)

```
Input:
    D: The initial damaged set of data items.
    BWS$_i$: The Blind Write set of a specific subgraph.
    CDS$_i$ :The Child Data Set of a specific subgraph.
    G: The data structure or graph representing the subgraphs.
    $t_a$: Attack time
    $t_{aff}$: Affected time
    $t_{last}$: last/final updated time
    Rl: released data item list that contains the released data items would be
kept after process
    PDl: potential damaged list
   damaged_data_items: A list of data items within the subgraph to retain
for further evaluation.
Procedure:
1.    Assess Subgraph for Damage considering each Gi:
   1.1.   If D ∩ BWS$_i$!=NULL for Gi
          1.1.1.   indicating there's at least one common data item.
          1.1.2.   Gi is damaged.
2.    Identify Data Items for Further Evaluation (if damaged):
   2.1.   If Gi is damaged:
          2.1.1.   For each data items y in CDS$_i$:
             2.1.1.1.       if y= f(z) where zϵ D or zϵ descendant of D
                2.1.1.1.1.          Add y to the PDl
          2.1.2.   For each data item x in PDI
             2.1.2.1.       Check table Final_updated_timetable
             2.1.2.2.       If t_last(x)=t_aff
                2.1.2.2.1.          Add x to the damaged_data_items list
             2.1.2.3.       Elif t_last(x)> t_aff
                2.1.2.3.1.          if the Gi = the subgraph containing the initial
                                    damaged data item
                   2.1.2.3.1.1.           check parents of x
                   2.1.2.3.1.2.           if x=f(z) where zϵD or zϵ descendant
                                          of D
                      2.1.2.3.1.2.1.         Add x to the damaged_data_items
                                             list
                   2.1.2.3.1.3.           Else
                      2.1.2.3.1.3.1.         Release x
                2.1.2.3.2.          Else
                   2.1.2.3.2.1.           Release x
3.    if D ∩ BWS$_i$ =NULL for G$_i$
   3.1.   Release all the data items in G$_i$
4.    Output Result:
   4.1.   return the damaged_data_items list for further evaluation.
```

Comment:

2.1.2.2. to 2.1.2.2.1.1: If the last update time is after the damaged time, only then it is checked if they belong to same graph or in the different graph. If they belong to same graph, then they could be affected depending on if one of there are damaged or not and if they belong to different graph then they can be released. This scenario can be explained in the following section.

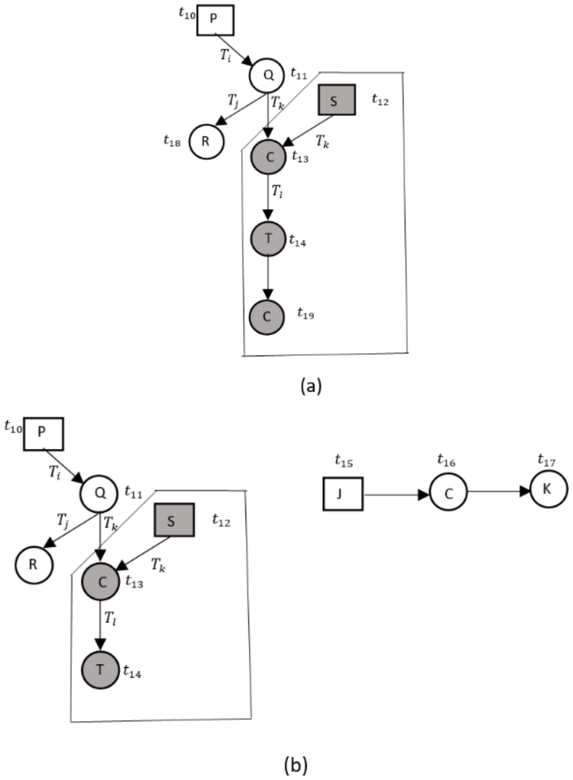

Figure 3. Multiple subgraphs in the data dependency (G).

TABLE III. FINAL UPDATED TIMETABLE (FOR SCENARIO (A))

Data Items	P	Q	S	T	***C***	R
$t_{\text{Last Updated}}$	t_{10}	t_{11}	t_{12}	t_{14}	t_{19}	t_{18}
Graph	G_2	G_2	G_2	G_2	G_2	G_2

TABLE IV. FINAL UPDATED TIMETABLE (FOR SCENARIO (B))

Data Items	P	Q	S	T	J	***C***	K	R
$t_{\text{Last Updated}}$	t_{10}	t_{11}	t_{12}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}
Graph	G_2	G_2	G_2	G_2	G_3	G_3	G_3	G_2

In Figure 3, two scenarios are discussed using G_2 and G_3. Tables III and IV display the final updated timetables for scenario (a) and (b), respectively.

When examining all the children for a particular graph, if it is discovered that the final updated time of a specific child is after the attack, the graph undergoes scrutiny. If the graph is distinct from the damaged graph, then the child data item is deemed safe for release. Because it belongs to a different graph that means it has no connection with the damaged items in the previous graph. Had there been a connection it would have been in the same graph. Since it is not in the same graph that guarantees there is no connection with any of the previously damaged values. However, if it belongs to the same damaged graph, it could be considered as damaged depending on its' parents. If it's one of the parents is damaged, then definitely that data item is damaged. And if none of its parents are damaged then it can be said that even after being damaged it recovered in the same graph. For example, in scenario (a), checking Table III reveals that the child data item *C* was finally updated at t_{19}, transpiring in the damaged graph G_2 (shaded part). Conversely, from the shaded area of Table IV (scenario (b)), even though *C* is a child of the initial damaged data item S, its last update occurred at t_{16} in the separate graph G_3, signifying that *C* is safe for release.

Our algorithm systematically processes each data item in the database to ascertain its affected status. If deemed affected, the data item is forwarded for recovery; if not, it is released. This comprehensive approach involves checking every graph, ensuring that each data item within that graph undergoes examination.

It is essential to note that there will be no data item existing outside of a graph. This assurance stems from the inherent nature of data item creation, where it is either generated blindly or based on another data item. In both scenarios, the data item is bound to be part of a graph.

As the algorithm meticulously examines each graph and subsequently categorizes every data item within as damaged or undamaged, the guarantee is established that the algorithm checks and classifies every data item as damaged or not damaged without exception.

VI. Simulation Results

In our simulation study, we consider five variables, which are as follows:

1. **Number of Transactions**: This represents the quantity of transactions executed per experiment.
2. **Number of Data Items**: Denotes the total count of data items utilized per experiment.
3. **Maximum Number of Operations per Transaction**: This parameter can vary and is randomly selected within the program.
4. **Maximum Write Operations**: Specifies the maximum number of write operations permitted per transaction, which can also vary.
5. **Number of Blind Writes**: Indicates the number of blind writes permitted in each experiment, calculated as 5% of the total number of transactions.

For consistency, we will maintain the following base values throughout the experiments:

- Number of Transactions = 200
- Number of Data Items = 1000
- Maximum Number of Operations per Transaction = 5
- Maximum Write Operations = 2
- Number of Blind Writes per Transaction = (Number of Transactions * 5%)

In each scenario, we will manipulate one variable while keeping the others constant. We will execute the program 25 times for each case and compute the average number of data readings using our blind writing method, as well as in normal transactions after identifying the malicious blind write.

A. Varying the number of transactions

In this scenario, we will be altering the number of transactions, ranging from 200 to 900, while maintaining the other variables (Number of data items, Maximum number of operations per transaction, Maximum write operations, Number of blind writes per transaction) constant.

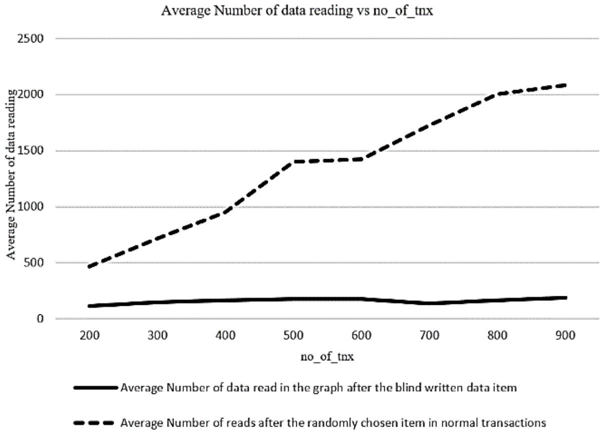

Figure 4: Varying the number of transactions.

As observed (Figure 4), when the number of transactions increases, the average data item reads after identifying malicious data in the usual log gradually rises. However, in our method, the average data item reads from the graph remains relatively constant but significantly lower compared to the usual scenario. This trend is attributed to the increasing number of transactions, which consequently leads to a higher number of blind writes and subsequently more graphs. Despite this, the average dependency per graph remains consistent. Hence, the graph representing our method appears almost flat due to this consistent average dependency per graph.

B. Varying the number of data items

In this scenario, we will be adjusting the number of data items, ranging from 500 to 3000, while keeping the other variables (Number of transactions, Maximum number of operations per transaction, Maximum write operations, Number of blind writes per transaction) constant.

In this scenario, we observe a significant reduction in the average reading of data items after identifying the damaged data in our method compared to the normal case (Figure 5). However, the graph remains relatively consistent. This consistency can be attributed to the fixed number of blind-written data items and the fixed number of written data items per transaction in our method. Since, the reading of data items is dependent on the data items written previously which means previously written data items are mostly read later on to write another data item, leading to consistent behavior even with variations in the number of data items.

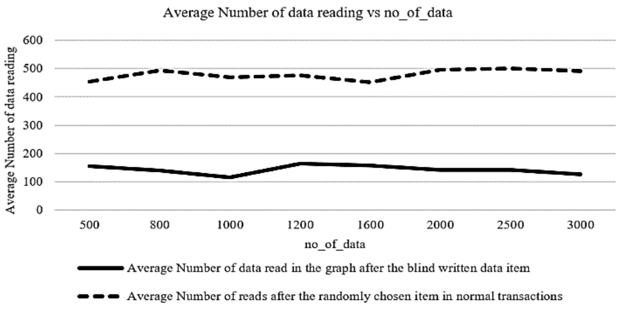

Figure 5: Varying the number of data items.

C. *Varying the Max number of operations per transaction*

In this scenario, we will be adjusting the maximum number of operations per transaction, ranging from 3 to 12, while maintaining the other variables (Number of transactions, Number of data items, Maximum write operations, Number of blind writes per transaction) constant.

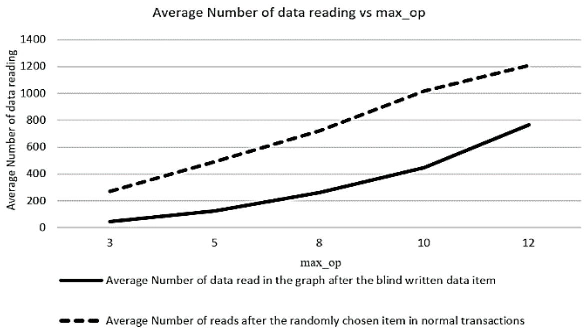

Figure 6: Varying the Max number of operations per transaction.

While both cases exhibit a gradual increase, the average read in our method remains significantly lower compared to normal transactions (Figure 6). However, the average read in our method increases gradually due to the higher number of operations per transaction. Since the number of write operations per transaction is fixed, more operations per transaction result in more read items, leading to increased dependency and consequently more data to read. This explains the gradual increase observed in the graph.

D. *Varying the Number of blind write per transaction*

In this scenario, we will be adjusting the number of blind writes per transaction, ranging from 1% to 10% of the number of transactions, while keeping the other variables (Number of transactions, Number of data items, Maximum number of operations per transaction, Maximum write operations) constant.

In this case, we observe a gradual decrease in the average reading in our method, while the average reading remains relatively constant in normal transactions (Figure 7). This difference can be attributed to the effect of varying the number of blind-written data items. In normal transactions, this variation has no impact. However, in our method, as the number of blind writes increases, the number of graphs also increases. Consequently, the number of data items depending on each graph decreases, leading to a decrease in the average reading.

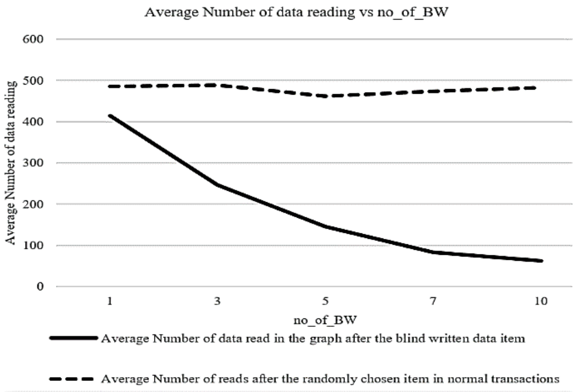

Figure 7: Varying the Number of blind writes per transaction.

It is important to note that in the first scenario where the number of transactions was varied, the graph representing our method remained constant. This was because the number of blind writes increased proportionally with the number of transactions. However, in the current scenario where the number of transactions and other factors are fixed, while the number of blind writes was varied, we observe a gradual decrease in the average number of data items read to recover after identifying the malicious data.

E. *Varying the Max write operations*

In this scenario, we will manipulate the number of maximum write operations, ranging from 1 to 5, while keeping the other variables constant (Number of transactions, Number of data items, Maximum number of operations per transaction, Number of blind write per transaction).

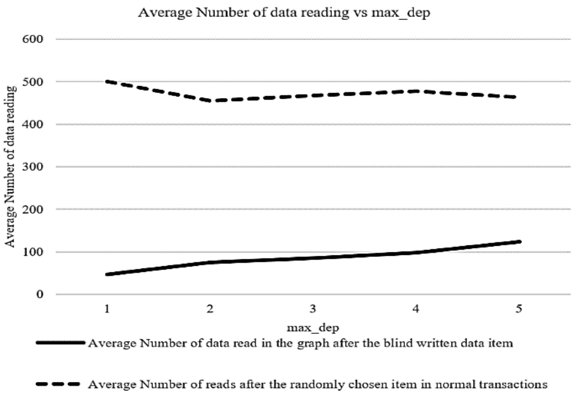

Figure 8: Varying the Max write operations.

In this case, it can be observed that in the normal case, the average reading remains somewhat constant (Figure 8). However, in our method, it increases gradually. This occurs because, with more write operations, the dependency also increases, given that blind writes are fixed in this scenario. Although blind writings are fixed, the process involves writing more data items after reading them, leading to increased dependency. Consequently, the graph shows a slight increase over time.

VII. CONCLUSION

This research proposes a novel technique for swiftly assessing damage caused by malicious attacks in fog computing systems. Traditional methods relying on log analysis are slow, hindering real-time data access. This model addresses this issue by leveraging blind write lineage, efficiently tracing the impact of blindly written data. The model constructs three key data structures during ongoing transactions: a Blind Data Set to track blindly written items, a Children Data Set to identify dependent data items, and Sub-dependency Graphs to represent intricate data relationships. When an attack is detected, the algorithm analyzes affected sub-dependency graphs and evaluates data items within them. This evaluation considers time parameters, release criteria, and potential damage to generate a final list of compromised data items. The simulation results show that the model offers advantages in speed, efficiency, and accuracy compared to traditional methods. However, applying this approach to real-world fog systems presents several requirements. These include the need for robust transaction logging, real-time dependency tracking mechanisms, and synchronization across distributed nodes. One key lesson learned is the critical role of data dependency management in preventing the propagation of damage. However, the diversity of fog systems introduces challenges, particularly the need to balance performance with accuracy in environments with heterogeneous node configurations and complex multipath dependencies. Future work will focus on refining the model to address attacks within specific time ranges, optimizing memory consumption through more efficient data structures, and ensuring scalability across diverse fog architectures. Additionally, exploring blockchain integration for immutable logging of transactions will further enhance the system’s security and resilience. Overall, this research offers a significant contribution towards building more robust fog computing systems capable of maintaining real-time data access and swift recovery in the face of cyberattacks.

ACKNOWLEDGMENT

This work has been supported in part by grant H98230-22-1-0321 issued by the National Security Agency as part of the National Centers of Academic Excellence in Cybersecurity's mission to expand cybersecurity research and education for the Nation.

REFERENCES

[1] M. S. Ahmad and B. Panda, "Damage Assessment in Fog Computing Systems: A Blind Write Lineage Approach." In 2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW), pp. 50-55. IEEE, 2024.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. "Fog computing and its role in the internet of things," In Proceedings of the first edition of the MCC workshop on Mobile cloud computing, pp. 13-16. 2012.

[3] C. Mouradian et al., “A Comprehensive Survey on Fog Computing: State-of-the-art and Research Challenges”, IEE Communications Surveys Tutorials, vol. 20, pp. 416-464. 2018.

[4] L. M. Vaquero and L. Rodero-Merino, “Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing”, ACM SIGCOMM Computer Communication Review, vol. 44., No. 5, pp. 27-32, 2014.

[5] G. Sun et al., “Security and privacy preservation in fog-based crowd sensing on the internet of vehicles,” Journal of Network and Computer Applications, vol. 134, pp. 89–99, 2019.

[6] M. Mukherjee et al., “Security and Privary in Fog Computing: Challenges”, IEEE Access, vol. 5, pp. 19293-19304, 2017.

[7] D. Wu and N. Ansari, "A Cooperative Computing Strategy for Blockchain-Secured Fog Computing," in IEEE Internet of Things Journal, vol. 7, no. 7, pp. 6603-6609, July 2020, doi: 10.1109/JIOT.2020.2974231.

[8] E. Viganò, M. Loi, and E. Yaghmaei, “Cybersecurity of Critical Infrastructure,” In The Ethics of Cybersecurity; Springer: Cham, Switzerland, 2020, pp. 157–177.

[9] P. Kotzanikolaou, M. Theoharidou, and D. Gritzalis, “Cascading Effects of Common-Cause Failures in Critical Infrastructures,” In: J. Butts and S. Shenoi (eds) Critical Infrastructure Protection VII. ICCIP 2013. IFIP Advances in Information and Communication Technology, vol 417. Springer, Berlin, Heidelberg. Communications, 2017, pp. 1-9.

[10] J. Ding, Y. Atif, S. F. Andler, B. Lindström, and M. Jeusfel, “CPS-based threat modeling for critical infrastructure protection,” ACM SIGMETRICS Performance Evaluation Review, 45(2), pp.129-132, 2017.

[11] D. Rehak, J. Markuci, M. Hromada, and K. Barcova, “Quantitative evaluation of the synergistic effects of failures in a critical infrastructure system,” International Journal of Critical Infrastructure Protection, *14*, pp.3-17, 2016.

[12] R. E. Stearns and D. J. Rosenkrantz, "Distributed Database Concurrency Controls using Before-values," Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, pp. 74-83, 1981.

[13] N. das Chagas Mendonca and R. de Oliveira Anido, "Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures," Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences, vol. 27. pp. 303–312, 1994.

[14] A. Burger, V. Kumar, and M. L. Hines, "Performance of Multiversion and Distributed Two-Phase Locking Concurrency Control Mechanisms in Distributed Databases," Inf. Sci., vol. 96, no. 1-2, pp. 129-152, 1997.

[15] P. Ammann, S. Jajodia and Peng Liu, "Recovery from malicious transactions," in IEEE Transactions on Knowledge and Data Engineering, vol. 14, no. 5, pp. 1167-1185, Sept.-Oct. 2002, doi: 10.1109/TKDE.2002.1033782.

[16] S. Tripathy and B. Panda, “Post-Intrusion Recovery Using Data Dependency Approach,” In Proceedings of the 2001 IEEE Workshop on Information Assurance and Security, pp. 156-160, 2001.

[17] B. Panda and P. Ragothaman, “Database Recovery in Information Warfare Scenario,” Handbooks in Information Systems, vol. 4, Information Assurance, Security and Privacy Services, pp. 73-97, H. Raghav Rao 12 and Shambhu Upadhyaya (Editors), Emerald Publications, United Kingdom, July 2009.

[18] R. A. Haraty, S. Kaddoura, and A.S. Zekri, “Recovery of business intelligence systems: Towards guaranteed continuity of patient centric healthcare systems through a matrix-based recovery approach,” Telematics and Informatics, 35(4), pp. 801-814, 2018.

[19] M. Alshehri et al., “A Novel Blockchain-based Encryption Model to Protect Fog Nodes from Behaviors of Malicious Nodes,” Electronics, vol. 10, pp. 313, 2022.