The Royal Split Paradigm: Real-Time Data Fragmentation and Distributed Networks for Data Loss Prevention

Jacob Matthew Hadden *jacob.hadden@gmail.com*
Texas A&M University – Corpus Christi
Corpus Christi, 78412, USA

Ahmed M. Mahdy *ahmed.mahdy@tamucc.edu*
Texas A&M University – Corpus Christi
Corpus Christi, 78412, USA

Abstract

With data encryption, access control, and monitoring technology, high profile data breaches still occur. To address this issue, this work focused on securing data at rest and data in motion by utilizing current distributed network technology in conjunction with a data fragmenting and defragmenting algorithm. Software prototyping was used to exhaustively test this new paradigm within the confines of the Defense Technology Experimental Research (DETER) virtual testbed. The virtual testbed was used to control all aspects within the testing network including: node population, topology, file size, and number of fragments. In each topology, and for each population size, different sized files were fragmented, distributed to nodes on the network, recovered, and defragmented. All of these tests were recorded and documented. The results produced by this prototype showed, with the max wait time removed, an average wait time of .0287 s/fragment and by increasing the number of fragments, N, the complexity, X, would increase as demonstrated in the formula: X = (.00287N!).

Keywords: Security, Data Loss Prevention, Distributed Networks, Networking, Virtual Testbed.

1. INTRODUCTION

There has been an increase in high profile data theft over the past few years. Both commercial enterprises and government entities appear to be vulnerable to data loss. Karen Kroll writes that data theft was the number one fear for corporate executives in 2013 [1] (p. 52). A review done by Scott Mace showed that some current software claiming to provide Data Loss Prevention (DLP) only provided Data Loss Detection (DLD) [2] (p. 48). Data security has several problems currently including: human susceptibility to social engineering, data must travel across the network, and every node on the network is a potential single point of failure. Despite these issues, there is little research in the field of DLP. The need for a secure method of storing and distributing data within a network that will thwart the majority of data theft techniques is the motivation behind this paper.

The inspiration for this paper came by equating a single piece of data to a royal family being attacked. When under attack, the royal family is capable of splitting up and sheltering in castles they believe are loyal. Even if some of the royal family members take shelter in a compromised location, the rest would be safe. In order for an attacker to succeed in usurping a royal family, the entire royal family must be accounted for. If even one member of the royal family escapes the attacker, there is a legitimate claim to the throne, and the attacker has only left a trail of evidence and wasted resources and has nothing to show for it. This “Royal Split” provides them with the best opportunity to preserve the royal family.

If a single piece of data is the royal family, the nodes on a network would be the loyal castles. Important data could split into fragments and reside on separate nodes, requiring an attacker to obtain all of the pieces before any whole data is risked. This paper develops, tests, and analyzes

this idea as a new paradigm for DLP. The purpose of this paper is to find a solution to the problems inherent in the current DLP paradigm. This paper is an attempt to provide a solution that addresses idea as a new paradigm for DLP. The purpose of this paper is to find a solution to the problems inherent in the current DLP paradigm. This paper is an attempt to provide a solution that addresses the current paradigm's security issues in a scalable manner, has limited human interaction, and can operate in real-time. This paper proposes a new paradigm for the storage and transfer of data, within an established network, with a focus on that data's security. The scope of this paradigm is limited to securing data-at-rest (DAR) and data-in-motion (DIM). The paradigm should be designed to have as small an attack surface as possible. This paradigm solution should incorporate data encryption, data fragmentation, and distributed technology, so that a single, fully intact, sensitive file does not exist on any single storage space. The objective of this paper is to provide a new paradigm for DLP that will provide a real-time solution that protects DAR and DIM.

This security paradigm adds layers of security on top of already established security measures, such as Advanced Encryption Standard (AES). A data file that is encrypted using AES 256 has a 1 in 9532 chance of being brute forced if the key is generated using an 8-bit character set. However, the complexity of an AES 256 encryption drops to a 1 in 1 chance of being broken if the encryption key is discovered. To add another layer of security, data files are fragmented and the fragments are distributed to nodes on a network. This layer of security can split the file between two fragments to as many fragment as there are bits in the file. If the file is not in plain text, or formatted in such a way that even while fragmented it is still vulnerable to breaches of confidentiality, the fragment must be put back in the correct order to be readable. For availability purposes the fragments will need to be duplicated and stored on separate nodes. This layer alone adds a minimum of a factorial complexity, assuming an attacker can differentiate one files fragments from another. The maximum number of fragments, of a single file, a node is allowed to have is all but one fragment. This is to prevent any single node from ever containing the complete set of fragments. As Table 1 below shows, the strength of this method grows rapidly with even a single node increase.

Fragments	2	3	4	5	6	7	8
Combinations	12	120	1,680	30,240	665,280	17,297,280	5.19E+8

TABLE 1: Number of combinations made possible by number of fragments.

The paradigm proposed in this paper focuses on securing DAR and DIM with as little human interaction as possible. This new paradigm contributes, to the fields of Cyber Security and Distributed Database (DDB), an innovative use of the fragmentation and defragmentation of data to be used in conjunction with DDB technology that provides a real-time solution for DLP problems occurring in the current paradigm. Furthermore, this paradigm offers a layer of security that offers a factorial security complexity and will be able to improve with the advancement of hardware

2. LITERATURE REVIEW

In pursuit of a new paradigm a review of the current research was done to establish what new, if any, techniques or methodologies could be leveraged to produce a more secure network for data. A review of the current tools and techniques used in network forensics that can validate the destination claiming to access the information is the actual location satisfies both the confidentiality and integrity of the network. An Input Debugging method requires an attack be detected before generating an attack signature to send to upstream routers, and the process is repeated upstream until the source is found or the search reaches the bounds of the Internet Service Provider (ISP) [4] (p. 18). This method may be suitable for tracking internal attacks, but has an expensive overhead cost of increase network traffic and specialized switches and routers. The input debugging method also requires an attack and this not acceptable in all environments. The controlled flooding method discussed, although it is cited as being "ingenious and pragmatic", is a self-inflicted denial-of-service (DOS) attack to find the location of a breach, so this method is

disregarded for that very reason [4] (p. 18). The internet control message protocol (ICMP) traceback, packet marking, and source path isolation engine (SPIE) methods all require that the entire path between the attacker and the victim have routers that are specifically setup to handle marking and requests for information [4] (p. 19-22). This requirement makes these methods fall outside the scope of most users, making them unlikely to be helpful in a scalable DLP paradigm. This research revealed a need for a solution that is less dependent on expensive equipment and utilizing what is available in as efficient a way as possible.

The commentary cited a paper submitted at a USENIX Symposium that called the scheme a phalanx and recommended using a good botnet to counter DOS attacks [5] (p. 53). A similar paper had theorized a way to use a botnet to buffer a DOS attack as well [6] (p. 45-48). Botnets are light weight programs that receive and send communication to a command node. The authors describe the botnets are capable of amplifying a small amount of data into a massive amount of data, but they are also able to condense. Condensed data can theoretically travel quicker through a network. A new paradigm in security could benefit from faster speeds, allowing more time for layers of security.

One paper points out that security policies fall short because sensitive information is not always declared under the right policy and keeping track of data requires significant overhead [7] (p. 53). The tracking of data is also overlooked when it comes to proper disposal policies for data [7] (p. 53). There is research being done to identify sensitive information, one group of researchers developed a way of identifying, with a 97% success rate, any document that contains sensitive data [8] (p. 20). The results of this research are successful enough that the scope of the new paradigm will be narrowed to only securing sensitive data. Their paper identified “3 types of data in an enterprise: data-at-rest, data-in-motion, and data-in-use” [8] (p. 21). As stated in the objective, the new paradigm will focus on securing DAR and DIM. The hope is that data-in-use (DIU) will be secure by focusing on securing DAR and DIM, in conjunction with whatever DIU security measures available.

The security requirements of confidentiality, integrity, and availability (CIA) are found in most security doctrine, but some also included authorization and authentication [9] (p 31). Confidentiality ensures data is only seen by those with the authority to see it. Integrity ensures data is not manipulated while in transit or without the authors knowledge or consent. Availability ensures data is always available to those who have permission to access it. Authentication “ensures the system knows the identities of all the entities interacting with it” [9] (p. 31). The proposed paradigm will support CIA completely, as well as Authentication, within the predefined scope.

To some “security is expensive and inconvenient” and because of this belief, and in some cases truth, it is often difficult to implement security measures that will be followed [10] (p. 655). Any new security measure that puts requirements on the user is unlikely to succeed due to users bypassing these requirements. If that security measure can operate without the user aware of its use then the user will not attempt, nor have the desire, to circumvent it. There is a scenario where there are multiple users on a network that are the “bosses” and “outrank” any IT worker attempting to enforce security requirements [10] (p. 656). This scenario is only problematic if the security measures in place causes the user to believe they are undergoing an unreasonable burden, real or imagined [10] (p. 656). However, this too can be resolved by incorporating an unobtrusive security method [10] (p. 656). Any paradigm created for the purpose of preventing data loss should be unobtrusive to the users on the network; i.e. it should be fast, and require little to no effort on the user to implement.

One paper’s definition of DLP lists the responsibilities as being able to “manage” “incident response” and should “enable corrective actions that remediate violations [11] (p. 12). A DLP should also be able to “discover” sensitive data anywhere on the network and “monitor” the data in all three states [11] (p. 12). A DLP should be able to “protect” that data, preventing it from being used in any way that could cause a security breach; e.g., printing, print screen, copy, etc. [11] (p.

12). The paper concludes that there is no “single effective solution” as of yet, revealing the need for such a solution to be designed [11] (p. 12). Although this paper purport that a solution should monitor all three states, the proposed solution will only focus on two as a layered approach may be the only way of developing a paradigm that is fully capable of meeting all of these responsibilities [11] (p.12).

3. RESULTS

The results showed a linear rate of growth in wait time as the number of fragments increased. The time to store data was always greater than the time to recover. With this information, the paradigm can be further enhanced by allowing the paradigm to determine the number of fragments based on the level of security and/or wait time that that is required and/or acceptable in the environment, with as little user interaction as possible.

The results contained some consistent outliers, where all the tests run for a particular split size of a file within a topology, as well as some inconsistent outliers, one or two wait times with more than a 1.5 times the standard deviation from the rest of the data for that sample set. The inconsistent outliers may have been bleed over from the rapid prototype developed services, program and test script, but this was not fully confirmed. The results were normalized by removing the max time from each set was removed from the analysis. The raw data results have been preserved, but unless it is expressly stated all data referred to in the results and conclusion are based on the adjusted results.

3.1 Five Node Topology

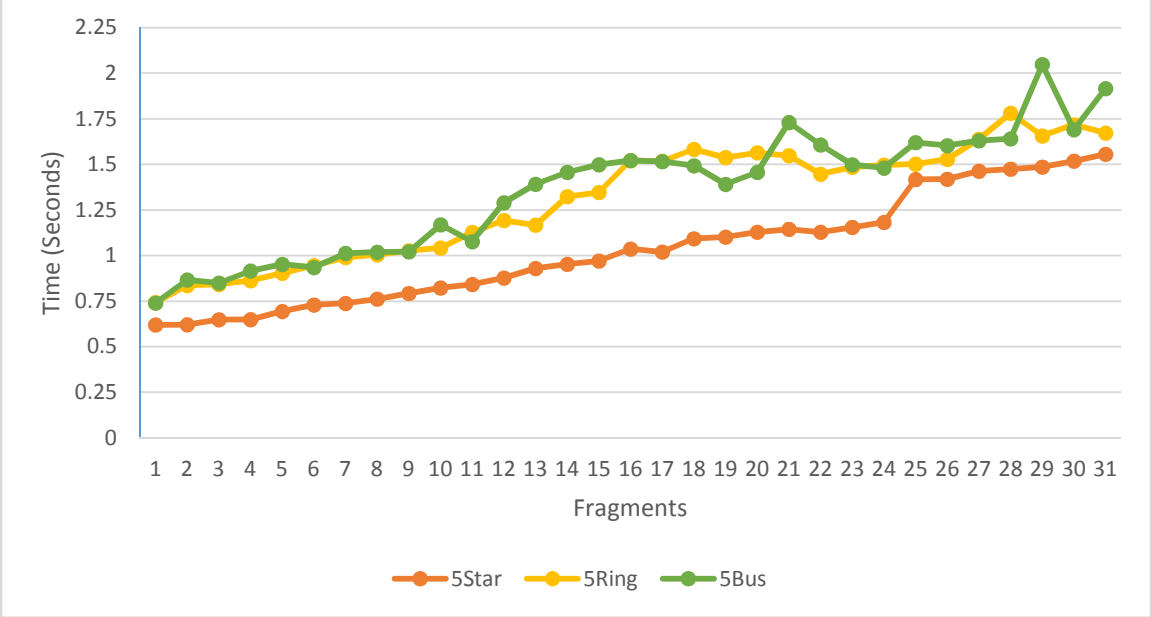

FIGURE 1: This figure shows the average retrieval wait time for a 10 MB pdf file, in a 5 node population topology.

Topologies appeared to play a nominal role in the efficiency of the paradigm. Although the Star topology appears to be faster in Figure 1, 5 node 10MB Retrieve above, the slope of this trial is only 32.2 ms/fragment compared to the Ring’s 32.8 ms/fragment slope, and the Bus’s 34.9 ms/slice. This trend carries over to the related store trial, as shown in Figure 2 below. The rate of growth for the Star topology was 35.4 ms/slice, the Ring’s 39.9 ms/slice, and the Bus’s 35.6 ms/slice. The results show that the store time is slower than the retrieve time. The results from

the test show a linear growth of wait time between 30 ms and 40 ms per number of fragments, regardless of the file size and node population.

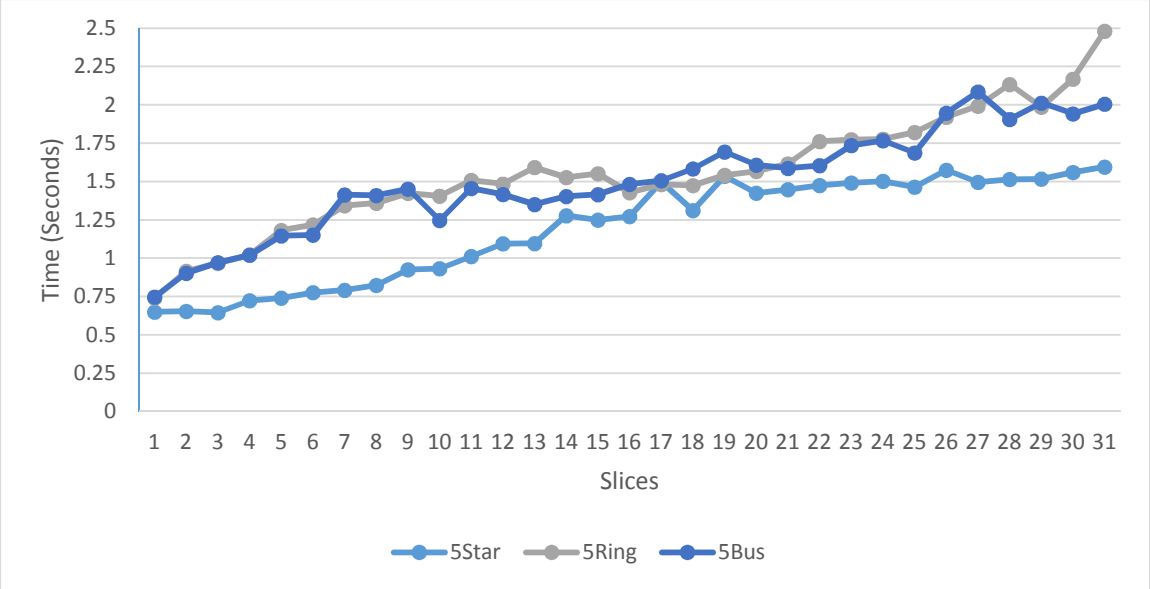

FIGURE 2: This figure shows the average storage time for a 10 MB pdf file, in a 5 node population topology.

3.2 Node Population

The same experiments were run in the same topologies, but with an increased population size from 5 nodes to 10. Population size reduced the wait time for storage and retrieval from the results of the 5 node related tests for the Bus and Ring topologies, but increased it for the Star topology. Although the Ring topology appears to be faster in Figure 3 below the slope of this trial is only 32.0 ms/fragment compared to the Bus's 36.8 ms/fragment slope, and the Star's 38.2 ms/slice.

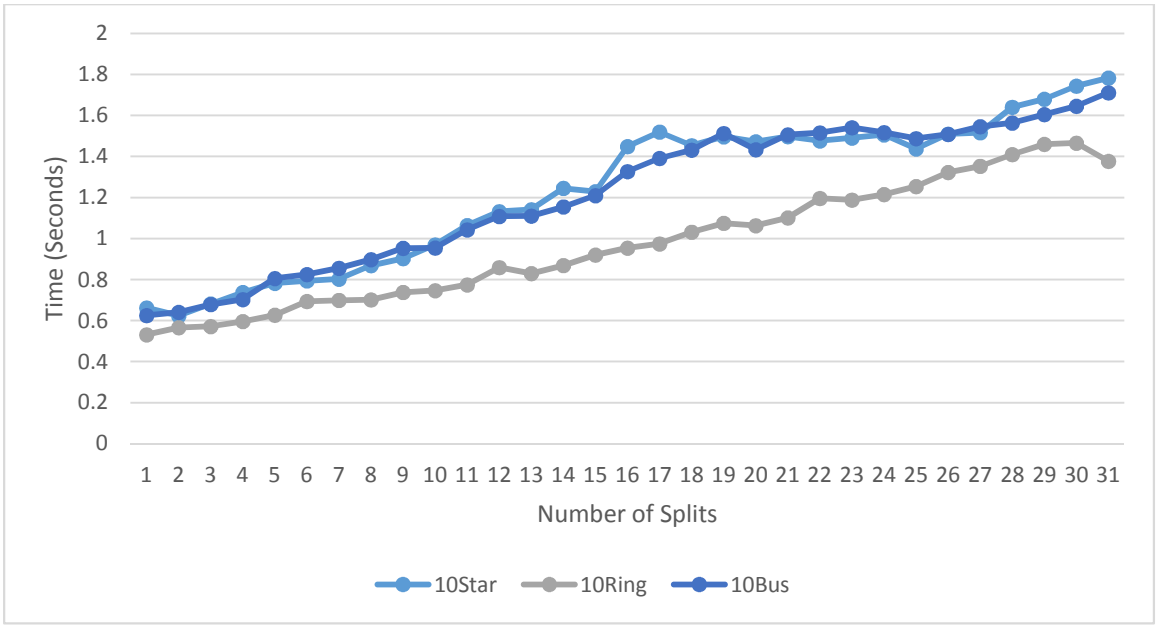

FIGURE 3: This figure shows the average storage time for a 10 MB pdf file, in a 10 node population topology.

This trend carries over to the related retrieval trials, as shown in Figure 4. The results show that the store time continues to be less than the retrieval time. The topologies are within 3 ms of each other in the retrieval growth rate: Star 32.2 ms/slice, Ring 32.8 ms/slice, and Bus 35.0 ms/slice.

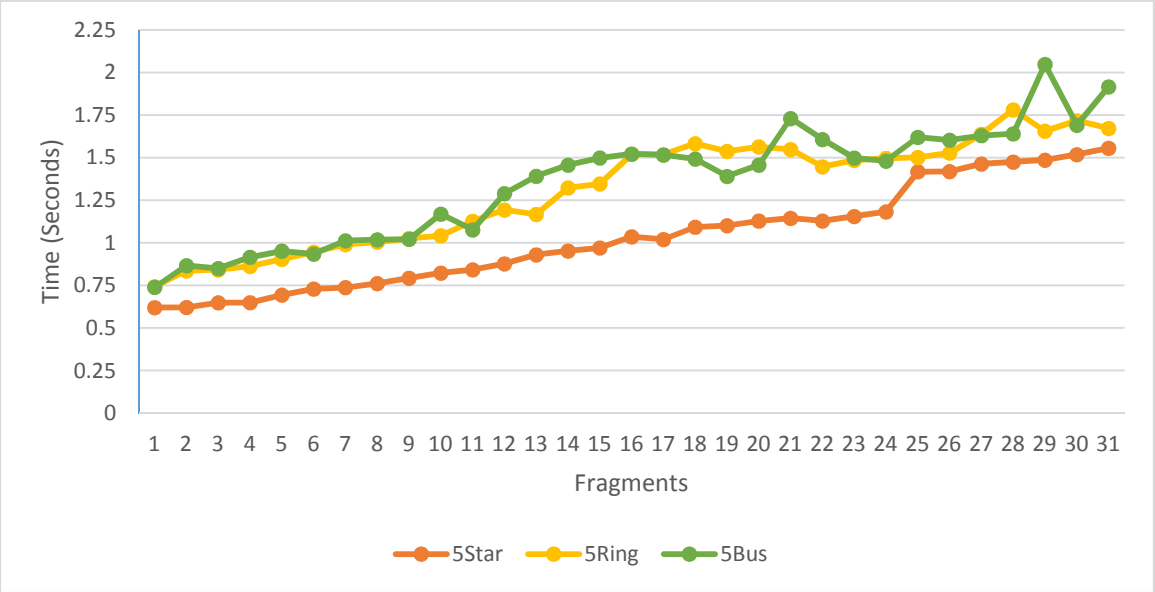

FIGURE 4: This figure shows the average retrieval time for a 10 MB pdf file, in a 10 node population topology.

3.3 File Size

The same experiments were run in the same topologies, with both a 5 and 10 node population size, but with an 8 KB pdf file and a 1 MB pdf file as well. The size of the file had a significant effect on the wait time for storage and retrieval of files in this paradigm. The topologies are within 3 ms of each other in the retrieval growth rate: Ring 28.8 ms/slice, Star 30.2 ms/slice, and Bus 30.3 ms/slice.

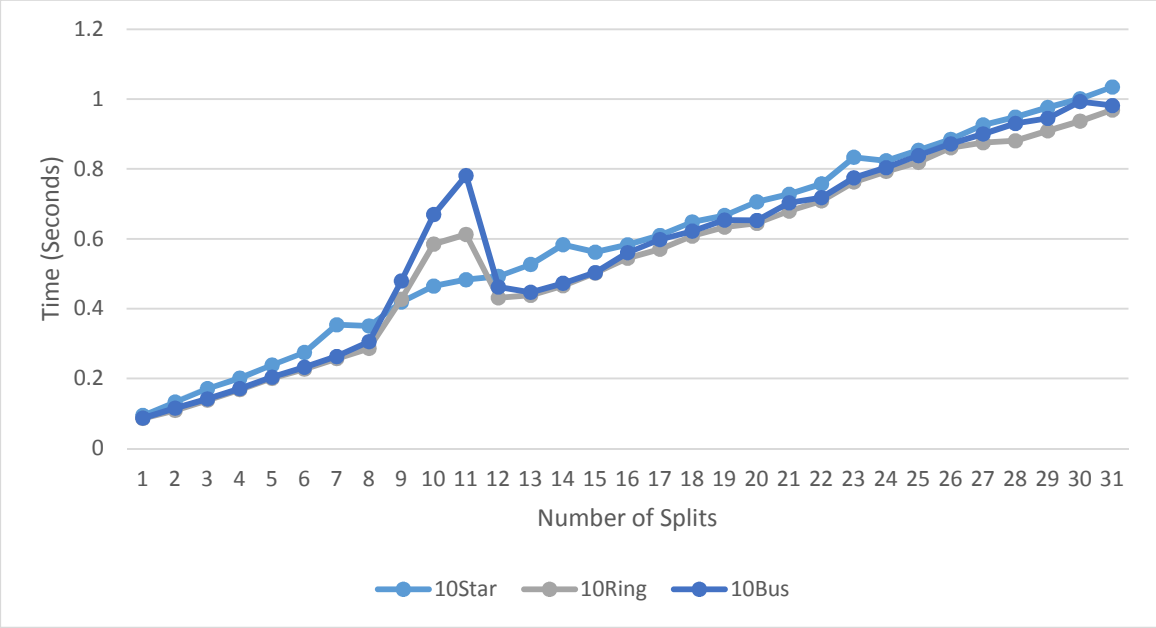

FIGURE 5: This figure shows the average storage time for a 1 MB pdf file, in a 10 node population topology.

This trend carries over to the related retrieval trials, as shown in Figure 6. The results show that the store time continues to be less than the retrieval time. The topologies are within 2 ms of each other in the retrieval growth rate: Ring 26.5 ms/slice, Bus 27.2 ms/slice, and Star 27.6 ms/slice.

FIGURE 6: This figure shows the average retrieval time for a 1 MB pdf file, in a 10 node population topology.

The size of the file has the biggest impact on the base wait time, as demonstrated from the 1 fragment line, which is the non-fragmented file store and retrieval time. The rate of growth is still less, as shown in Figures 5, 6, 7, and 8. In Figure 7 it shows the smallest file size tested, an 8 KB file. For storage the rate of growth was: Ring 28.9 ms/slice, Bus 29.7 ms/slice, and Star 30.2 ms/slice.

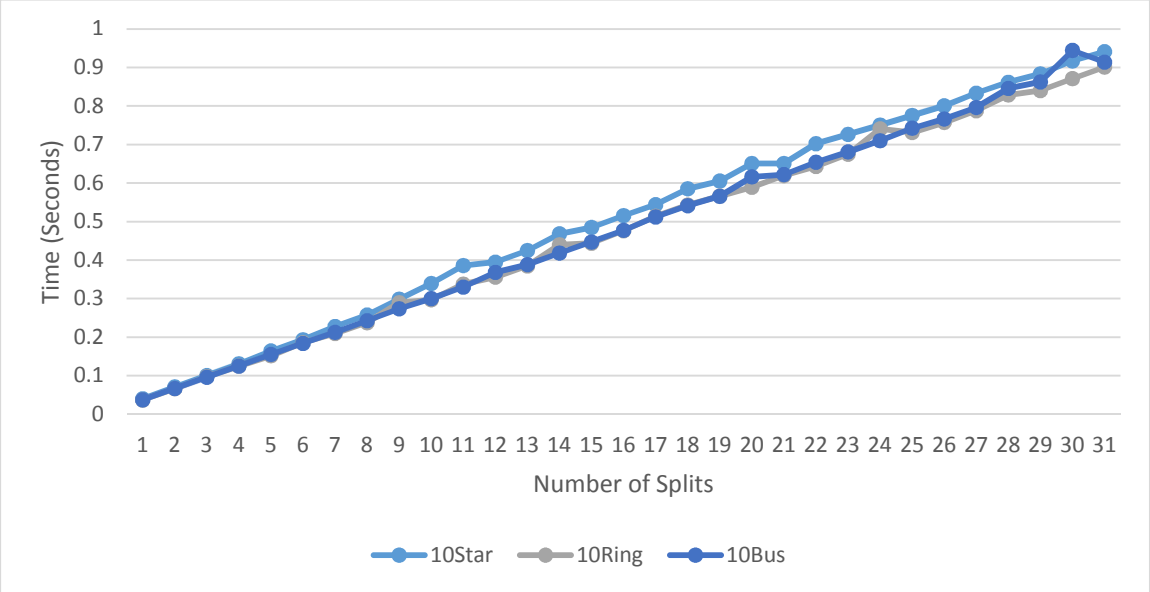

FIGURE 7: This figure shows the average storage time for an 8 KB pdf file, in a 10 node population topology.

In Figure 8 it shows the smallest file size tested, an 8 KB file. For storage the rate of growth was: Ring 26.9 ms/slice, Bus 27.0 ms/slice, and Star 27.5 ms/slice. The baseline storage and retrieval time for the file is less than that of the 1 MB or 10 MB file times.

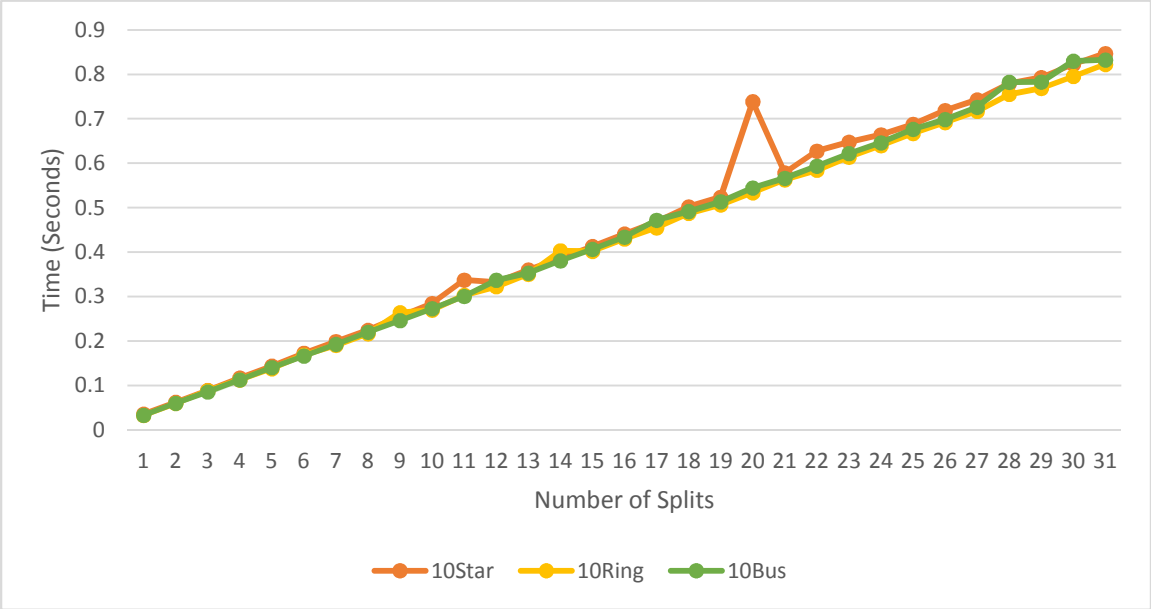

FIGURE 7: This figure shows the average retrieval time for an 8 KB pdf file, in a 10 node population topology.

4. DISCUSSION

This paradigm's focus on securing DAR and DIM required a solution that was capable of being usable without providing a noticeable burden on users. A study done for web page retrieval by Nah found that the average person is willing to wait "about 2 s for simple information retrieval tasks on the web" [3] (p. 160). The maximum wait time recorded, including the discarded maximums, was 8.50 s and occurred during a 10 MB trial, in a 5 node Bus topology, where the file was fragmented 27 times. The maximum average wait time of any of the retrieves was 2.63 s in a 10 MB trial, in a 5 Ring topology, where the file was fragmented 31 times. The average additional wait time for all of the trials done was an extra 28.7 ms/slice. The prototype for this paradigm is capable of providing an extra N factorial complexity for an additional 28.7N ms of wait time, where N is the number of fragments. If an attacker had the exact fragments needed for a single file and was able to go through ten permutations a second, the cost-to-benefits ratio of an average case scenario can be calculated as:

$$X = 28.7msN!/2,$$

where X is the time, in ms, required to go through the combinations before finding the correct combination. If the file was also encrypted, then the file would still need to be decrypted after it was put back together. Table 2 shows the average additional wait time per fragmentation, as well as the security it would provide if an attacker attempted to put it back together, should they already have the correct pieces.

Fragments	Average Case	Worst Case	Incurred Wait Time
2	2.87E-03 s	5.74E-03 s	28.7 ms
3	8.61E-03 s	1.72E-02 s	57.4 ms
4	3.44E-02 s	6.89E-02 s	86.1 ms
5	.172 s	.344 s	114.8 ms
6	1.03 s	2.07 s	143.5 ms

7	4 s	8 s	172.2 ms
8	33.5 s	67 s	200.9 ms
9	5 m	10 m	229.6 ms
10	50 m	100 m	258.3 ms
11	9 hrs	18 hrs	287.0 ms
12	4.5 days	9days	315.7 ms
13	60 days	120 days	344.4 ms
14	2 years	4 years	373.1 ms
15	3 decades	6 decades	401.8 ms
16	5.5 centuries	11 centuries	430.5 ms
17	9 millennia	18 millennia	459.2 ms
18	169.5 millennia	339 millennia	487.9 ms
19	3223 millennia	6446 millennia	516.6 ms
20	6.45E+04 millennia	1.29E+05 millennia	545.3 ms
21	1.35E+06 millennia	2.71E+06 millennia	574.0 ms
22	2.98E+07 millennia	5.96E+07 millennia	602.7 ms
23	6.85E+08 millennia	1.37E+09 millennia	631.4 ms
24	1.64E+10 millennia	3.29E+10 millennia	660.1 ms
25	4.11E+11 millennia	8.22E+11 millennia	688.8 ms
26	1.07E+13 millennia	2.14E+13 millennia	717.5 ms
27	2.89E+14 millennia	5.77E+14 millennia	746.2 ms
28	8.08E+15 millennia	1.62E+16 millennia	774.9 ms
29	2.34E+17 millennia	4.69E+17 millennia	803.6 ms
30	7.03E+18 millennia	1.41E+19 millennia	832.3 ms
31	2.18E+20 millennia	4.36E+20 millennia	861.0 ms

TABLE 2: Additional wait time per fragmentation, and the time it takes to brute-force putting the file back together at 10 permutations/s.

As shown in Table 2, this paradigm is capable of providing a significant amount of security without imposing a noticeable burden to the users. Layered security is capable in this paradigm, as it does not care what the file type is, encrypted files can be stored in this fashion and retrieved. This paradigm is also capable of improving with technology. These tests were done with current processors and ram speeds, along with virtual network speeds. As these devices increase in speed and efficiency, so too will the paradigm.

5. MATERIALS AND METHODS

The following sections describes the testing environment, program design, and implementation.

5.1 The Design

The design phase of this paper sets the scope and expectations this new paradigm will operate under. The minimum requirements expected for the testing phase to succeed are chosen for a reasonable starting point. Based on the success or failure of experimentation, the final minimum requirements proposed in the conclusion will be adjusted. The theoretical setup for the environment used to create test scenarios will be used for proof of concept, as well as for fine tuning the minimum requirements.

The new paradigm this paper proposes is narrow in scope and will be able to incorporate the positive aspects of the current paradigm, without including the negatives. This new way of thinking focuses on securing DAR and DIM, in real-time, using distributed networks, a central coordinating node, and the fragmentation of data, with limited human interaction. It is assumed that encryption and AC are used in the new paradigm. DAR is secured in this paradigm because it is fragmented into N-Number of “fragments” and each fragment is stored onto a node in a distributed network. DIM is secured in this paradigm because it travels in a fragmented form within a secure tunnel when distributed from the Central Data Controller (CDC) to a distributed node (DN). DIM is secured when travelling between the client node and the CDC by a secure tunnel using direct communication.

The paradigm acknowledges the CDC and direct communication with it is a single point of failure. This is mitigated by utilizing proper network security. All nodes on the network should be able to communicate to the CDC with direct point-to-point (PTP) communication. This is achievable with modern networking techniques and hardware, and therefore will not be focused on in this paper. The CDC's attack surface is reduced to physical access and three ports, which can be reduced to two after the DN's have been added to the CDC's list. The first port is for initial node connection; this port can be disabled once all node have connected, preventing un-approved nodes from joining. When a node joins, it sends the size of the allocated space; this is added to the CDC's list so that files are not sent to nodes that do not have space for them. The second port is for communication initiated by the controller to the DN's. This communication will be limited to the commands "Store", "Retrieve", and "Delete". The store command sends a fragment of data from the controller. The retrieve command sends the name of the file to retrieve. The delete command sends the name of the file to delete. The third port is for communication with the client program. The client program can store, retrieve, or delete a file. When the client program sends a file to be saved, it includes the split size. The retrieve command sends only the file name of the file to be retrieved.

The known vulnerabilities for this new paradigm are as follows. DAR could potentially be stolen in this paradigm if all of the nodes that were part of the distributed network were compromised and the attacker was able to collect all the fragments of data. If all of the fragments were collected, the attacker would still need to sort the fragments into the correct files and, then, organize them in the correct order. Without adding duplication for redundancy, this would require M number of fragments be sorted into an unknown G number of groups, and then, the unknown N number of fragments within the groups be sorted in the correct order. If there were only 10 files distributed on the network and each only had 10 fragments, there would be 100 pieces and no way for an attacker of knowing how many files there were or how many pieces each file was split into. If the attacker did know the number of files and the number of splits there are:

$$X = (10!)/[10!(100-10)!]$$

or 17,310,309,456,440 different 10 fragment combinations would exist for the attacker to work through, and only 1 of those are valid.

5.2 The Test Environment

To reduce variability all nodes on the network will be created on the cyber Defense Technology Experimental Research (DETER) testbed. The DETER testbed is a remotely accessible virtual environment that allows for the rapid deployment of entire networks. This allows for tests to be done in a stable environment with tight variable control. The DETER testbed lives on two clusters that are tunneled together, one at USC Information Sciences Institute and the other at UC Berkley. Using the DETER technology, all virtual environment used in this paper will be use 10 MB bandwidth for all connections. All nodes will have the Kali Linux distro available on the DETER testbed. Each node will have Python 2.7 installed and run the same version of RPyC. Simulations will be done with two different population sizes, five nodes and ten nodes, arranged in three different topologies: bus, ring, and star. All nodes, except for the CDC, will be a part of the DN Network. The Python random number generator will perform a modulus equation with the number of nodes in the distributed network to decide which node will receive the next split file. All tests will be assessed for time, the starting time begins after the client command is sent from the user and ends when the response that the task is complete is returned to the client program from the command server. All six environments will test the run times for storing and retrieving data. Three files will be used in these environments: a 7,945 byte (8KB) pdf sample file, a 1040920 byte (1MB) pdf file, a 10,467,710 byte (10MB) pdf sample file, and a 1 GB zip file will be attempted. The three files will be stored and retrieved thirty-one times in all six environments at split sizes between one and thirty-one.

After the tests are completed the results will be analyzed. The data will be checked to determine if the paradigm provides predictable save and retrieve times based on file size, the number of

splits, topology, or the number of nodes. Analysis will also be done for determining a range of optimal split sizes given the size of the file, topology and number of nodes.

5.3 The Implementation

The following paragraphs detail the implementation of the services and programs used in the test environment. These services and programs were created using a rapid prototype methodology and were subsequently developed with a proof-of-concept design. The prototype results will be analyzed acknowledging that these results may be improved upon with better design. For testing purposes, the RPyC sockets and connections were hard coded into the Client Program and the CDC so that rapid testing could be implemented. The node services were programed with the CDC destination hard coded as well.

5.4 The CDC

The CDC service needs to be setup prior to any other part of the system. The service requires Python 2.7 and RPyC to operate. It communicates using three different ports. One listens for requests to join the distributed network. One is used for communication to DN's, this port is only activated when sending commands to the DN's. The third port actively listens for requests from a client program to store or retrieve files. When a request to store a file is retrieved, it expects a split size.

The CDC maintains a list of all nodes connected to the distributed network. When data is sent to the CDC to be stored, the file is divided, on the bit level, by the split size. This number is stored and is used in a for loop to read in only that number of bytes to a temporary file. This temporary file is then sent to a random node in the distributed network. Python's Random Number Generator is used to select which node in the list will retrieve the data. The CDC names each fragment and stores it in a heap, these names are randomized to obfuscate the connection to the original file name.

When the CDC is sent a request to retrieve a file, it is given the filename. If the filename matches one in the list of stored files, it finds the fragment names in the heap and sends the request to the corresponding DN's that hold the data. As the fragments arrive the CDC appends them together in the correct order and returns it to the node that made the request with the Client program.

5.5 The DN's

The DN's must be setup after the CDC is setup, but before the Client Program can run. The DN's prototype only joins the distributed network, it does not send the allocated space to the CDC. The DN's waits for commands from the CDC only. It stores files sent to it with the names given to it, and returns files back to the CDC when the filename is sent to it with the retrieve request. The prototype DN Service can be found in Appendix D.

5.6 The Client Program

The Client Program is a program that takes in two commands: store and retrieve. The store command expects two arguments: the filename and the split size. For the prototype, the filename must correspond to a file in the same directory. The retrieve command expects one argument: the filename. Both commands are sent to the CDC and await a response. The store command receives an affirmative message from the CDC and displays the message back to the user. The file that was sent to the CDC is deleted on the Client computer. The retrieve command receives the file and an affirmative message from the CDC and displays the message for the user.

5.7 The Tests

The environments were tested individually on the DETER testbed. The services had to be setup manually, so that they were started in the correct order. Once the CDC and the DN's were running and connected the client node was prepped for the tests. The tests were done within the TEMP directory of the Linux environment. All four of the test files were copied to the TEMP directory, so the results could be compared to the originals. A batch script was created to automate the testing. The batch script made a fresh copy of the original test file for each run, ran

31 trials for each split size between 1 and 31 for all 4 test files. The test script eventually had to be altered to exclude the testing of the 1GB file, because it would not complete before the virtual environment timed out. The test script outputted the run times for each trial to a CSV file sorted by topology, network size, and file size. These files were imported into an Excel spread sheet and were analyzed. The results of the data showed that there is a linear increase to the wait time for storing and retrieving data.

This guideline is used for all journals. These are the manuscript preparation guidelines used as a standard template for all journal submissions. Author must follow these instructions while preparing/modifying these guidelines. This guideline is used for all journals. This guideline is used for all journals. These are the manuscript preparation guidelines used as a standard template for all journal submissions. Author must follow these instructions while preparing/modifying these guidelines. This guideline is used for all journals. This guideline is used for all journals. These are the manuscript preparation guidelines used as a standard template for all journal submissions. Author must follow these instructions while preparing/modifying these guidelines. This guideline is used for all journals.

6. CONCLUSION AND FUTURE WORK

The results from this work have laid the ground work for new research in the fields of DLP, Distributed Systems, and Cyber Security. Specific to advancing the proposed paradigm, the development of an efficient CDC and DN service, as well as a Client program. The code used is in the Appendix. The CDC should include an algorithm to predict the optimal split size for any given file. This would reduce the attack surface area by removing this option from the Client Program and the User's input. Research on how long the average person is willing to wait for vital data would be needed for a maximum wait time to be incorporated into the optimization algorithm.

The rate of growth has an average of 28.75 ms, with a 2.5 ms standard deviation, to set a base line for the paradigm, the CDC could use down time to generate file of a specific size and test a "single" split. The CDC could store this data for use in the optimization algorithm. This test data could be used to systematically create dummy split files on the DN Network which would further enhance the security of the paradigm. As processor speed, network speed, and read/write speed increase in the future an artificial intelligence (AI) algorithm to continually optimize the paradigm would allow the paradigm to update itself without the need of human interaction.

Acknowledgments
This work has been supported, in part, by National Science Foundation grant CNS-1042341.

7. REFERENCES

1. Kroll, K. Crafting an effective data security policy. *Compliance Week*, 10(110), 2013, 52-53.

2. Mace, S. Options in Data-Loss Prevention. *Healthleaders Magazine*, 15(11), 2012, 44-48.

3. Nah, F. F. (2004). A study on tolerable waiting time: how long are Web users willing to wait? *Behaviour & Information Technology*, 23(3), 2004, 153-163.

4. Meghanathan, N., Allam, S. R., & Moore, L. A. Tools and techniques for network forensics. In S. Fischer-Hubner & N. Hopper (Eds.), Privacy Enhancing Technologies (18-37). 2009, Canada: Springer-Verlag Berlin-Heidelberg.

5. Rapoza, J. (2008, May 5). Botnets vs. botnets. *eWeek*. p. 53.

6. Dixon, C., Anderson, T. E., & Krishnamurthy, A. (2008). Phalanx: Withstanding multimillion-node botnets. In NSDI, 8, 45-5.

7. Kroll, K. (2013). Crafting an effective data security policy. Compliance Week, 10(110), 52-53.

8. Hart, M., Manadhata, P., & Johnson, R. (2011, January). Text classification for data loss prevention. In Privacy Enhancing Technologies (pp. 18-37). Springer Berlin Heidelberg.

9. Venkatasubramanian, K. K. (2009). Security solutions for cyber-physical systems (Doctoral dissertation, Arizona State University).

10. Ezekiel, A. W. (2013). Hackers, spies, and stolen secrets: Protecting law firms from data theft. Harv. J. Law & Tec, 26, 649-695.

11. Liu, S., & Kuhn, R. (2010). Data loss prevention. IT professional, 12(2), 10-13.