DelphiFL: An investigation into Attacks and Defenses for Federated Learning

Jonathan Flores[1,*], Erin Kendall[2], Adam Crayton[1], and Hailey Whipple[3]

[1]Computer Science, Boise State University, University Drive, Boise, 83725, Idaho, USA
[2]Computer Science, Transylvania University, Broadway, Lexington, 40508, Kentucky, USA
[3]Computer Science, Utah Tech University, University Ave, St. George, 84770, Utah, USA
[*]Corresponding author: jonathanflores@u.boisestate.edu

Abstract

As technology and machine learning evolves and changes, developers have been met with the ever increasing need to maximize the utility of their product while still ensuring privacy and security for its users. To do this, researchers have proposed a machine learning paradigm where each client maintains their own local model that learns from their data. After that, the model is sent over a network to a centralized server to be aggregated into a global model. This process is known as Federated Learning (FL). Although FL is useful for protecting private data, since it is dependent on user contributions, its security suffers. In particular, poisoning attacks, in which a malicious client aims to further a secondary objective by changing the data before the model is trained, have been found to be effective against the process of Federated Learning. Our research is twofold: we aimed to create several destructive poisoning attacks and an aggregation method, named DelphiFL, that utilizes the Zero-Trust policy and leverages existing methods in a process known as method chaining. To this end, we tested both standard defenses and attacks, and the ones we created against one another.

Keywords: Machine Learning, Artificial Intelligence, Federated Learning, Zero-Trust, Method Chaining, Security.

1. Introduction

The average person has most likely had their information stolen over the internet. There have been measures presented to the public that people can follow to further protect their data, but it is not enough. Significant effort has gone into solving this on-going concern, and developers have turned their attention to Artificial Intelligence (AI) and

Machine Learning (ML) as possible answers. As a result, Federated Learning (FL), a distributed machine learning paradigm, was invented.

Federated Learning is a machine learning setting where multiple entities (also called clients) collaborate in solving a machine learning problem, under the coordination of a central server (Bhagoji et al., 2019; Lyu et al., 2022; Ma et al., 2023). However, FL models are not immune to attacks. Of the various kinds, one of the most detrimental is known as a poisoning attack. This type of attack aims to compromise the systems' robustness by manipulating the data that the local models learn from (Bagdasaryan et al., 2020; Bhagoji et al., 2019; Lyu et al., 2022; Zhao et al., 2020). There have been standard methods created in order to defend against each of these different poisoning attacks, but each one individually is unable to defend against all attacks. One method some have used to defend against these attacks involves incorporating the zero-trust model into their system, which is a policy framework within machine learning in which clients start on a no-trust basis (Vucovich et al., 2024). As a result, in order to further privacy and security for Federated Learning, we look into both creating new poisoning attacks, as well as improving defensive strategies for FL. As such, we created DelphiFL, which uses the zero-trust policy and a "Unified Federation" method that is robust against various threats.

We begin this paper by reviewing prior research done into both Federated Learning and poisoning attacks. Next, we explain our research methods and go over our defenses and attacks. Finally, we conclude by presenting our results and noting future directions for this research.

2. Background

2.1 Byzantine General's Problem

When it comes to the internet, many people are concerned with their information being stolen. An example of this is credit card information when making an online purchase. Once this information is input into a textbox of some sort on a web page or application, it is sent to the server. However, the medium from client to server, also known as the network, is the greatest vulnerability when it comes to the internet.

An early well-known concept for distributed systems, introduced by Leslie Lamport, is the Byzantine Generals' Problem, also known as the Two Generals' Problem. Leslie described the situation as an abstraction where there is a group of generals camped with their troops around an enemy city (Lamport et al., 1982). As seen in Figure 2.1, Lieutenant 1 is sending a message to Lieutenant 2. This message can contain any sort of information, for example, to attack the enemy or retreat. The enemy can notice this message and intercept communication at any given time. Changing the message or using the information given to their advantage is a known problem in Computer Science. Leslie has created this abstraction for conflicting information in computer systems, but it is also used when communicating over a network.

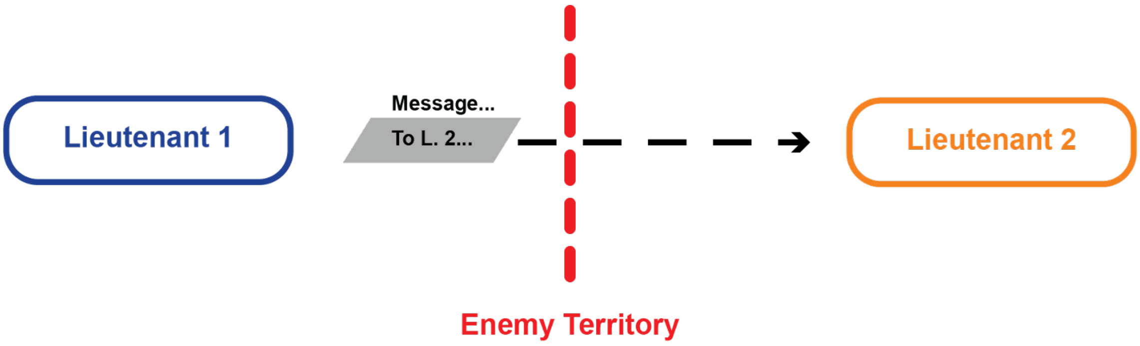

Figure 2.1: Visualization of the Two Generals Problem

The Byzantine Generals' Problem can also be applied to the security of private data. For example, each client has its own raw data and that data is eventually sent to the server to be utilized. However, the network is considered to be "enemy territory" and the data being sent over it is the message from client (Lieutenant 1) to server (Lieutenant 2). This poses a security risk and violation of privacy, since that message can contain private data unique to that client. Solutions to this problem include adding either encryption or "noise" to the data (Ma et al., 2023). For example, instead of using all of the private data in a program, one might only take a section of it or "blurr" the information to protect users' data. This is not enough though, since threat actors, or the "enemy," are able to find ways of decrypting the message and stealing user data.

2.2 Privacy and Utility

An example of sending data from a client to the server would be a web application. A user may want to login to their account on that application, and in order to do so, the server must know that information. For the server to have that information, the user must sign up for an account, and that private data must be sent from the client to the server to be logged. After that, the user is able to login to their account, but every time they do so, they are sending their credentials to be compared with the ones saved in the server, which is considered validation.

The example of a web application shows the way that data can be utilized to provide better security, however, there is always the concern about violating users' privacy. This is considered a fundamental relationship, or a curve, between privacy and utility (Bonawitz et al., 2021), and is demonstrated here in figure 2.2.

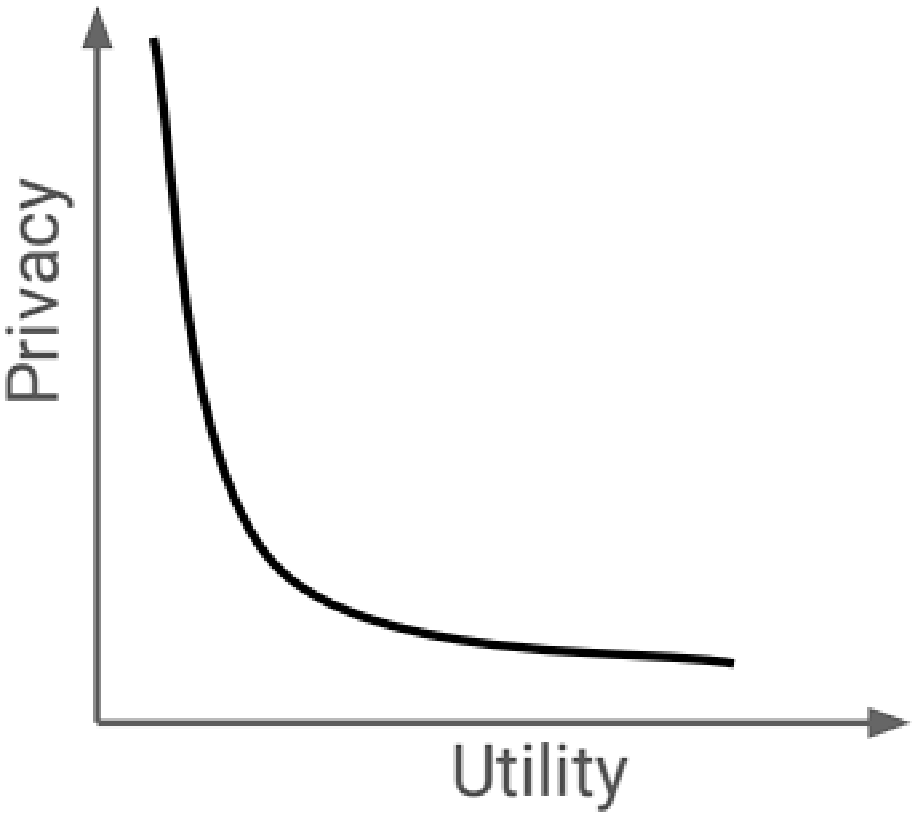

Figure 2.2: Privacy vs. Utility Graph [Bonawitz et al., 2022]

In other words, as utility increases, privacy decreases. Utility can increase by sharing more data from the client to the server, and although this can be helpful in certain ways, it violates privacy. The other way around, by increasing privacy, utility decreases due to there being less information shared to the server that it can work with. This is also a concern because of the potential for the client to be malicious. One of the goals of current computer science researchers is to shift the current paradigm, demonstrated here in figure 2.3, up and to the right to increase utility at at more meager cost to privacy.

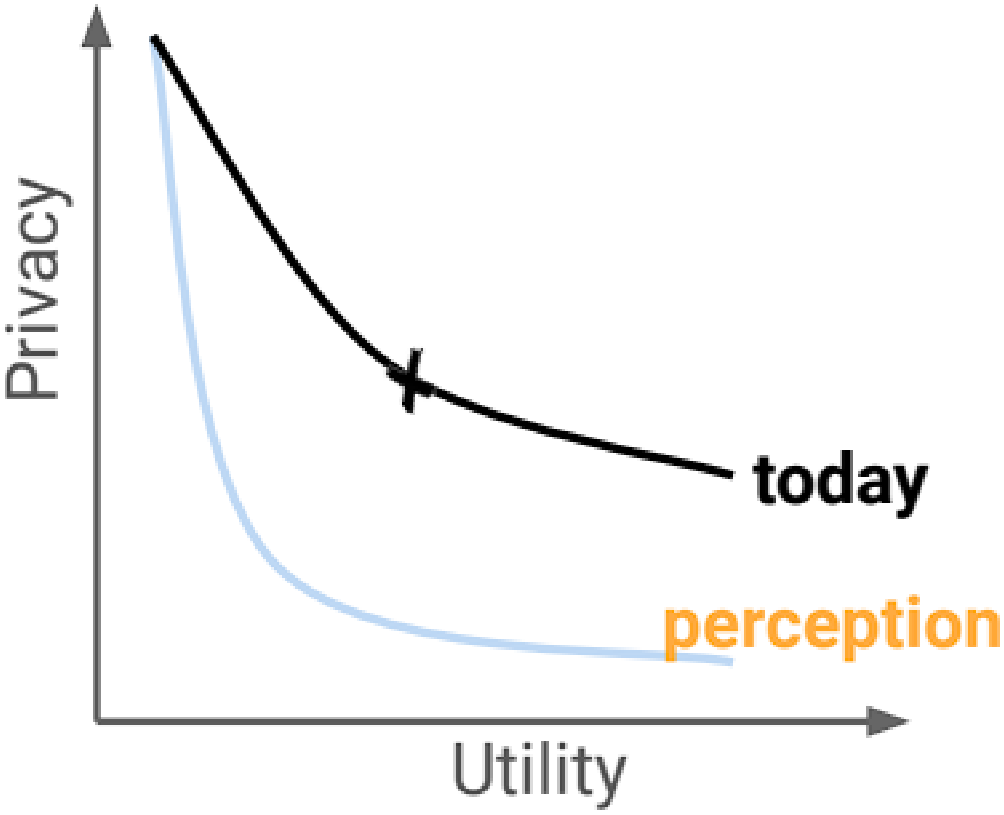

Figure 2.3: Privacy vs. Utility Perception [Bonawitz et al., 2022]

2.3 Federated Learning

As a result of these concerns, Federated Learning (FL) was invented. Federated Learning is used in a client-server model, where each client has their own local Machine Learning (ML) model that is training off of encrypted raw data (Bonawitz et al., 2021). As seen in Figure 2.4, the local model from the clients is uploaded to the server. After that, the local model parameters are aggregated together to form a global model. The process repeats with the server sending each client the global model.

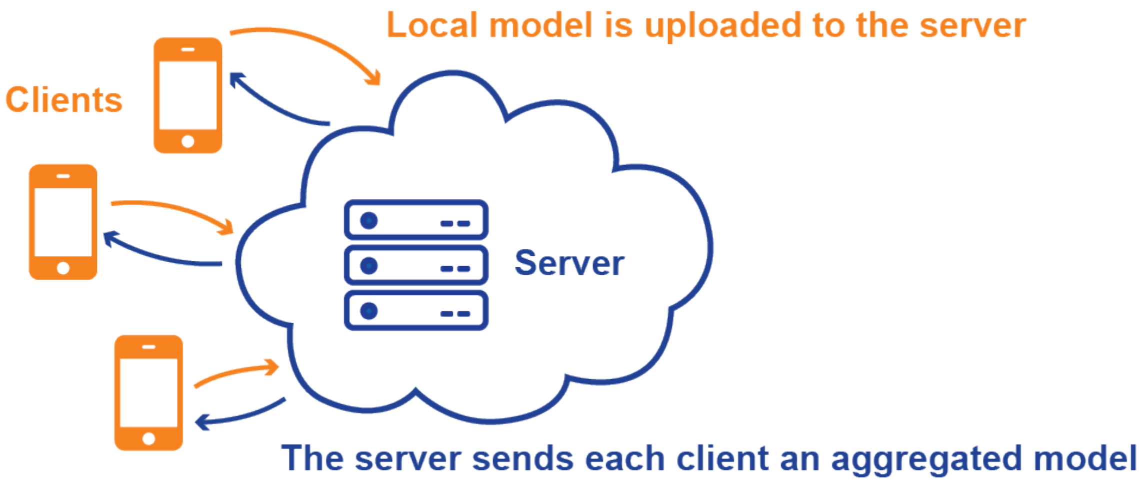

Figure 2.4: Visualization of Federated Learning

In other words, FL adds an extra security measure that methods without the use of ML do not have. However, Federated Learning too has its vulnerabilities and problems. Since FL is reliant on user contributions, malicious clients, also known as threat actors, can change the raw data before it is trained upon. This is known as a poisoning attack, which can damage the global model later on depending on the amount of threat actors that there are (Ma et al., 2023).

2.4 Zero-Trust

Zero-Trust is a framework in which it is assumed that any client could be a threat actor, and, as such, all clients begin being equally weighted within an FL system (Vucovich et al., 2024). As time progresses, some users will be given a higher influence on the global model, or in other words, higher trust, due to the consistency and uniformity of their data contributions. As such, the possible impact of threat actors regularly making malicious contributions is minimized due to the decreasing weight attributed to their contributions.

3. Related Works

Significant research has been made towards Federated Learning for both attacks and defenses. However, there is still more research needed, and the work done by Huang et al. (2023) was instrumental in this experimentation. They established in their work, "A Federated Learning for Generalization, Robustness, Fairness: A Survey and Benchmark," a simulation method that was altered and utilized within this work.

3.1 Prior Defenses

Fang et al. (2020) noted that there have been many different proposed aggregation rules, such as mean, Krum, trimmed mean, and median. They are the most common aggregation rules, along with Bulyan, which Mhamdi et al. (2018) proposed in order to counteract some of the failings of Krum. Each of these has different strengths, but their respective weaknesses have led to many researchers attempting to find better alternatives. In particular, while Fang et al. (2020) found Krum was useful for mitigating Byzantine attacks, multiple studies have found that it does not work to prevent poisoning attacks, and some even mention it could potentially cause more problems than it solves (Bagdasaryan et al., 2020; Bhagoji et al., 2019; Wu et al., 2024). With this in mind, one proposed alternative to Krum is a zero-trust policy. Vucovich et al. (2024) created a Federated Learning system that incorporated the zero-trust policy, named FedBayes. Unfortunately, they decided to use global mean and standard deviation for their aggregation rules, reducing the model's robustness, and they were not thorough in documenting their process, which made verifying their results difficult.

3.2 Prior Attacks

Jagielski et al. (2018) also researched poisoning attacks as they relate to regression learning, and potential defenses against those attacks. Other researchers, such as: Bagdasaryan et al. (2020), Bhagohi et al. (2019), and Ma et al. (2023) who worked with FL generally, and Zhao et al. (2020) whose research focused on using Generative Adversarial Networks to launch effective attacks with less information, all looked into implementing poisoning attacks in Federated Learning. Thus, several different poisoning attacks have been proposed. Wu et al. (2024) described many different ones, including label flipping, where one label is switched with another, Bagdasaryan et al. (2020) presented the idea of a backdoor attack, where a threat actor would add a subtask to the model, and they described in detail how to accomplish such an attack in a FL setting. Additionally, Gupta et al. (2023) proposed an attack which involves inverting the loss values of FL so that they diverge instead of converge, reducing the overall accuracy of the model.

4. Methodology

4.1 Adversarial Approach and Structure

This project was organized into an adversarial structure in which two researchers (Jonathan and Hailey, hereby known as Blue Team) were tasked with creating a more robust defensive system for an FL model from the scope of global aggregation methods, and two researchers (Adam and Erin, hereby known as Red Team) were tasked with creating more damaging attacks on an FL model. The goal of this adversarial structure was to broaden the scope of the project to answer the questions of what makes a more robust FL system and what weaknesses exist and can be exploited in FL itself. The project was divided into several rounds alternating between development stages and conflict stages with a constant literature review throughout. This cycle of alternating stages was terminated when either of two criteria were met: 1) Blue team would need to broaden the scope of their methods beyond that of just global aggregation in order to defend against an attack or 2) Red Team could no longer find attacks that fared better than those established in the MARS paper. Furthermore, it is of note that the Red Team was allowed white box access to not only the simulation code, but also any code developed by the Blue Team. This was done to simulate a worst case scenario in attacks on FL; one in which the attacker knows everything about the model.

4.2 Testing and Simulation

Models were evaluated using the benchmarking software developed by the MARS group at Wuhan University [Huang et al.]. This software provided accuracy measures on predictions, measures of attack success rate on backdoor attacks, as well as various customization options on the simulations themselves including: threat actor rate, noise tolerance, and data set. Simulations were run on two datasets: MNIST and USPS. MNIST and USPS are datasets consisting of handwritten digits used in a classifier. The training size of MNIST was 60,000 images and the test size was 10,000 while the training size for USPS was 7,291 images and the test size was 2,007. Furthermore, attacks were run with a range of 10% to 30% threat actors out of a simulated pool of 10 clients. Additionally, the attacks tested were those provided in the MARS paper, including two backdoor attacks (attacks intended to maintain accuracy while also accomplishing a second, undesired task): a standard backdoor attack (Base Backdoor), and a more targeted backdoor attack (Semantic Backdoor). In addition to this, two Byzantine attacks, which are attacks designed to lower prediction accuracy, were tested: PairFlip and Random Noise. It is of note that, due to time restrictions, there were several other Byzantine attacks that were tested on MNIST at 20% threat actor rate that were cropped out for further testing. Finally, several attacks of various types designed by the Red Team were tested. For backdoor style attacks, the local method was maintained as the standard FedAVG algorithm, and the metric used for comparison was the mean attack success rate, which measured how many images were mislabeled in the way that the attack was designed to achieve. For byzantine style attacks, the mean accuracy drop was the metric used for comparison and this was measured by comparing the mean accuracy of the predictions, when set the FedProx local method, to those of the FedProx global method combined with the FedProx local method with no attack present, also known as the benign.

5. Defenses

5.1 Blue Team Methodology

The Blue Team approach to creating a more robust defense for FL models was heavily reliant on a process known as Method Chaining. This process of Method Chaining arose naturally through investigation and experimentation, and it was later brought to the attention of the researchers that this process was already proposed in [Ponte et al., 2024], but was not explored. Method Chaining, in this context, is the process of taking existing security measures and algorithms and "chaining" them together: running them in sequence and passing the results of one method to the next. This resulted in a model that is more resilient to attack. Method Chaining was used by our team in an attempt to create a "unified federation method," [Huang et al., 2023], that is more robust to a broader range of attacks as many current security methods for FL exist; however, they each only tend to focus on a single type of attack. The results of this Method Chaining are what we have deemed DelphiFLV1 and DelphiFLV2. The major components of DelphiFLV1 are described below.

5.2 DelphiFLV1

DelphiFLV1 consists largely of a Bulyan structure, which is an application of a Krum procedure to select a representative mean distribution of model parameters to be used as a comparative base. Krum refers to the process of selecting a mean or otherwise representative distribution through the calculation of the standard deviations and means of existing gradients over a set of gradients. [Blanchard et al., 2017] After this Krum procedure, outliers are trimmed from the top 20% and bottom 20% of the data as determined by the error of the parameters obtained through a comparison of the distributions to the Krum selected mean distribution. DelphiFLV1 differs from a standard Bulyan structure in that it uses a median instead of a mean in the Krum process, and in that it incorporates a zero-trust policy to determine the weights of the remaining, untrimmed local models in the global aggregation based on the similarity of the gradients to the previous global model update. Zero-trust is a concept in which we assume that all clients are equally likely to be a threat actor, and as such, no single client should be trusted to submit a completely honest model, and, as such, the submitted local models should have their influence on the global model weighted according to some other criterion. In the case of DelphiFLV1, as mentioned earlier, this criterion is the similarity of the gradients to the previous global model using the norm of the distributions when represented as a matrix. The baseline model is a model trained in an environment where the security of the data can be guaranteed.

5.3 DelphiFLV2

DelphiFLV2 was created as a response to a particularly effective backdoor attack developed by the Red Team. This method differs from V1 in that it takes the conditional trimming one step further and adds an RLR component to the overall method structure. RLR, or Robust Learning Rate, is a method proposed by [M. S. Ozdayi, M. Kantarcioglu, and Y. R. Gel, 2024] in which the learning rate of the model is altered dynamically based on, "the sign information of agents' updates." In essence, the step size of the optimization algorithm being used in the model is altered based on specific descriptors of the data calculated from the models themselves.

5.4 Strategies of Note

Furthermore, two strategies of note within the scope of this project include the FedAVG and FedProx methods with both local and global components. We used these two strategies as both baselines for comparison and the local methods for our own original code. FedAVG is a fairly standard method of creating and aggregating models through the use of averaging. In this sense, it is the "base" method within FL. As such, because the Blue Team focused on global aggregation methods and not local ones, FedAVG was used as the local method for all backdoor attacks. FedProx is a method that demonstrates notable resistance to Byzantine style attacks. FedProx achieves this by essentially mimicking FedAVG, but adjusted for heterogeneous data sets. As such, FedProx served not only as our benign baseline for comparing the success of byzantine style attacks, but its local method served as the base for all tests against them as well, including DelphiFLV1 and DelphiFLV2 for the sake of fair comparison. Using FedAVG as the local method for backdoor attacks and FedProx as the local method for Byzantine attacks is the procedure established by [Huang et al., 2023].

6. Attacks

We were able to run a variety of attacks on all the different defenses and different datasets. This first set of attacks are those built-in to the simulation and had existed for some time. For this reason, most recent developments in defenses have focused on mitigating these types of attacks. We used these as the baselines with which to compare our new attacks.

6.1 Provided Attacks

Of the Byzantine attacks included in the simulation, we used Pair Flip and Random Noise for all of our tests. Pair Flip essentially flipped the pairs of any given neural net, while Random Noise shuffled the neural nets of the various models. We were also able to use two backdoor attacks, Base Backdoor and Semantic Backdoor. The adversary in a backdoor attack ensures that the model continues to perform well on its intended task, but also performs with high confidence on a subtask that will allow the adversary to gain some control over the model. One common example of this is in a word prediction algorithm, where the attacker guarantees that specific sentences end a certain way, potentially with the intention of influencing a person's opinions on a topic. (Bagdasaryan et al., 2020; Wu et al., 2024). The first attack, Base Backdoor, added a specific series of red pixels to the upper left hand corner of the image, labeling it as a category of the attacker's choice.

For example, in our simulation, we ensured that all of the poisoned data was labeled "2". Because of this, in the final model, whenever an image contains those pixels, no matter what the number in the image is, it will be labeled as a 2. In contrast, in Semantic Backdoor, images with one label are lightly scrambled, then relabeled to a category of the attacker's choice. In our simulation, we swapped images labeled "3" to be labeled "2". Therefore, in the final model, images that could be dubiously considered 3 are instead labeled 2.

6.2 Implemented Attacks

The second set of attacks are ones that were not originally part of the simulation, either because they were originally proposed in a different paper or because they are entirely new. For the attacks we created ourselves, we built on top of the backdoor attack code already in use in the simulation, creating new functions that would replace those for base backdoor and semantic backdoor. Sneaky Backdoor is one such attack. It is similar in theory to semantic backdoor: the images with one label are scrambled and relabeled, but the scrambling step is significantly more intense. For example, in our simulation, images originally labeled as "3" were scrambled beyond recognition and labeled as "2". In effect, this means in the final model, any image which does not look like a number will be labeled 2.

All of the attacks titled "sneaky random" served us as a baseline for which types of modifications and disguise techniques worked and which were caught by the methods. Sneaky Random works by taking a certain percentage of the images a malicious client has equal to the noise rate and scrambles them. For instance, if the noise rate is 0.5, then 50% of the images that any given malicious client has will be scrambled. The goal of this attack is that if the number of scrambled images are under the noise rate, it will be more difficult for the FL method to identify the bad clients. Sneaky Random 2 scrambles all of the images a malicious client has in accordance with the noise rate. As an example, if the noise rate is 0.5, the image tensors will be multiplied by random numbers that are between 0 and 0.5. The purpose of this attack is the same as the first sneaky random, but instead of reducing the number of scrambled images, it reduces how much the images are scrambled in the first place. Sneaky Random 3 is similar to the first sneaky random, but instead of just shuffling the images, it also shuffles the targets. While changing both the image and the target makes the attack easier to detect, the goal is to cause more damage in the process. Sneaky Random 4 is also similar to the first sneaky random in that it only takes a certain percentage of images of a malicious client that is equal to the noise rate, but rather than scrambling the images, it assigns each image a random label. Sneaky Random 5 is different from the previous sneaky random attacks in that it shuffles the images in a different way. Like the second sneaky random attack, it creates a new tensor with random numbers less than the noise rate, but rather than multiplying the tensors, it instead adds them. Because of this, it causes less damage to the image itself, as well as a mathematically different “type” of damage. The goal of this particular attack was to see whether it was easier or more difficult to detect a shuffled image if the shuffling was done by addition rather than multiplication.

In addition to creating attacks that modified the images strictly using math, we also wanted an attack that modified them in a concrete visual manner. Gauss Images utilizes the properties of the images themselves in order to create an attack that is, in theory, harder to detect. We created a new image that was purely gaussian noise, then overlaid that image on top of the original. After that, the target was also randomized. In theory, this would mean that each image was more difficult to identify, and giving them random labels would muddy the data pool in general. The goal of all of this was to make it more difficult to distinguish between all numbers, therefore making the model less accurate.

Inverted Gradient aims to find a mapping of labels for poisoning the data which would move the gradients of the clients participating in a federated training in an opposite direction (Gupta et al., 2023). This is based on an idea called “Anti-Training.” It trains a machine learning model using an inverted loss function where at every iteration, instead of converging toward the minima (the wanted loss), it will produce gradients that diverge away from the minima. This algorithm is demonstrated in Figure 6.1.

Algorithm 1: Inverted Gradient Function

```
Function: Inverse_loss(target, prediction)
    criterion = torch.nn.CrossEntropyLoss()
    loss = criterion(target, prediction)
    inv_loss = 0
    if loss < 0.001 then
        loss = 0.001
    inv_loss = 1 / loss
return inv_loss
```

Figure 6.1: Inverse Loss Algorithm [Gupta et al., 2023]

The inverse loss is found through an *inverse_loss* function, where it takes the initial loss (L_1) of the model, and replaces it with an inverse of that initial loss (L_2). Once the *inv_loss* has been calculated and returned, the overall model will use that as its loss, causing the model to converge towards a completely wrong value. In Torch, the function *torch.nn.CrossEntropyLoss()*, gets the cross-entropy *loss* of the model, where $0 < loss < 1$. Ultimately, this means that $inv\_loss \geq 1000$ since we are dividing 1 by *loss*. The main takeaway from this attack, is it changes the mathematical calculation on what the models were going to be trained off of.

Atropos is an attack experimenting with the idea of combining multiple attacks together. In particular, it combined Inverted Gradient, Sneaky Backdoor, and Gauss Images: randomizing the images like in sneaky backdoor, then overlaying them with an image of Gaussian noise, all while inverting the loss functions in the background. The goal of this attack was to find out whether it was more or less effective to perform multiple attacks at once, and if it was possible to create an attack that would both serve to implement a backdoor as well as reduce the accuracy of the overall model.

7. Results

As stated in the methods section, backdoor attacks and byzantine attacks are measured using different metrics. Backdoor attacks are measured based on their "attack success rate" (how often compromised data is successfully labeled according to the backdoor algorithm), and byzantine attacks are measured based on their "mean difference from benign" (the difference between the mean benign accuracy and the mean attacked accuracy, which can also be viewed as a drop in overall accuracy).

7.1 Benign Results

Before we began running each attack against each model, we had to get a mean accuracy of each model with no attacks being executed. We ran DelphiFL V1 & V2 with both FedProx and FedAVG. DelphiFL used FedProx when we were executing byzantine attacks against the model, and FedAVG when we were executing backdoor attacks. Through this, we were able to get our benign percentages. For MNIST, we got a 98.8% accuracy for our benign test. For USPS, we got 96.26% as our accuracy for our benign test. These values helped us determine the metrics we used to measure Byzantine attacks.

7.2 FedProx/FedAVG – 20% BCR Results

Backdoor				
	MNIST		USPS	
Attack Type	Mean Accuracy	Mean Attack Success Rate	Mean Accuracy	Mean Attack Success Rate
Base	98.34	88.64	92.96	11.44
Semantic	98.38	1.58	94.16	2.65
Sneaky	98.34	95.76	94.67	87.36
Atropos	98.68	89.66	93.79	86.72
Byzantine				
Attack Type	Mean Accuracy	Mean Difference from Benign	Mean Accuracy	Mean Difference from Benign
Pair Flip	98.68	0.12	94.99	1.27
Sym Flip	98.57	0.23	94.45	1.81
Random Noise	98.11	0.69	87.81	8.45
Lie Attack	98.3	0.5	85.44	10.82
Min Max	98.13	0.67	85.47	10.79
Min Sum	98.23	0.57	85.64	10.62
Sneaky Random 1	98.67	0.13	94.81	1.45
Sneaky Random 2	98.68	0.12	94.63	1.63
Sneaky Random 3	98.11	0.69	94.01	2.25
Sneaky Random 4	97.25	1.55	92.49	3.77
Sneaky Random 5	98.73	0.07	94.52	1.74
Inverted Gradient	98.61	0.19	94.53	1.73
Gauss Images	97.79	1.01	93.27	2.99
Atropos	98.68	0.12	93.79	2.47

Table 1: 20% BCR FedProx/FedAVG Data

7.2.1 Backdoor Attacks

Each of these backdoor attacks have their own special task that they are trying to accomplish. Base backdoor adds red pixels in the top left of each compromised image, whereas semantic backdoor, sneaky backdoor, and Atropos randomize the pixels inside each compromised image (with sneaky backdoor and Atropos randomizing more drastically/intensely than semantic).

As we expected, semantic backdoor had the lowest attack success rate, as it did not randomize many of the pixels inside each image. Because of this, it was easy to find which image had been attacked and which had not. As shown in Table 1, only 1.58% of the compromised images in the MNIST dataset were successfully used, and only 2.65% of the compromised images in USPS were successfully used.

Both sneaky backdoor and Atropos, however, had high success rates for both datasets. The reason behind this is that sneaky backdoor has much more stealth implemented into the attack (indicated by the name). However, this stealth was not of a traditional sense, it came in the form of the attack randomizing more pixels than semantic backdoor does, and to a greater degree. Counterintuitively, this was indeed stealthier, because now any image that was altered or otherwise difficult to classify was automatically mis-categorized in the manner that the attack algorithm dictated. This is why the percentages for each are so high in Table 1 (95.79% for MNIST and 87.36% for USPS). Atropos uses sneaky backdoor in its combined attacks. Unfortunately the other attacks make it easier to detect, so the defenses caught more of the compromised data. The interesting one out of this bunch is the base backdoor attack. In Table 1, the mean attack success rate in MNIST is 88.64%, whereas in USPS, it drops all the way down to 11.44%. One of the reasons why we believe base backdoor was not nearly as successful in USPS is because of the size of the dataset. Since there was a larger test and training size in MNIST, it had a higher chance to allow compromised images to be mis-categorized. In contrast, USPS had a much smaller dataset and was more precise, so when there was a small change, like red pixels in the top left corner of the image, there was a good chance the model would notice it and not accept it.

7.2.2 Byzantine Attacks

Byzantine attacks, as stated previously, are designed to decrease the accuracy of the model itself, causing misclassifications and causing the training to not work as it is intended.

As shown in Table 1, for USPS, each byzantine attack caused the accuracy of the model to lower by at least 1% on the FedAvg/FedProx models, with some of these attacks causing the accuracy to lower by almost 11%. The attacks appearing to have the most damage were the Lie Attack, Min Max Attack, and Min Sum attack. Notably, the Lie Attack appeared to cause the most damage, having a 10.82% difference from the benign model. The other two attacks were not far away, with Min Sum having a 10.62% difference and Min Max having a 10.79% difference.

USPS overall, however, seemed to be more susceptible to byzantine attacks than it was to backdoor attacks. This caused most of the attacks to be anywhere in between the 1% to 4% range, with some outliers having even larger differences than that. However, if we look at FedProx/FedAVG in the MNIST dataset, almost all of the attacks had a very minimal difference, with the highest of them all being Sneaky Random 4, with a 1.55% difference. Because Sneaky Random 4 changed the labels rather than the images themselves, it had a higher chance to cause more damage to the accuracy. However, 1.55% is still not a very large difference and would not call immediate attention to the attack. The only other notable attack is Gauss Images, which has a 1.01% difference. With the ability to add noise to the images, it allowed the training to not be as accurate in its labeling, causing the overall accuracy to decrease very slightly.

The MNIST dataset appeared to have a much better handle on byzantine attacks compared to the USPS dataset. A reason why we believe this is the case is due to the sizes of the datasets. Being that USPS has a smaller dataset, it is harder to use more images to have a more accurate training. This would mean that USPS will use the compromised values, which will cause the accuracy to decrease overall. MNIST has a much larger dataset, allowing it to use other images to compare and contrast compromised images. It has a better time detecting compromised images and data values, so not as many will be used. This caused the overall accuracy to stay intact and remain relatively close to the benign value.

The models themselves work differently against each type of attack depending on what the dataset is. For USPS, it appears to mitigate backdoor attacks quite well, whereas it is very susceptible to byzantine attacks. Using MNIST, it is the exact opposite. Backdoor attacks appear to be much more successful, whereas byzantine attacks do very little damage to the accuracy of the model.

7.3 DelphiFL V1 – 20% BCR Results

DelphiFL was the new model that we developed. Its goal was to successfully increase security, while also keeping up the robustness and utility of the entire training.

Backdoor				
	MNIST		USPS	
Attack Type	**Mean Accuracy**	**Mean Attack Success Rate**	**Mean Accuracy**	**Mean Attack Success Rate**
Base	97.75	10.25	87.65	10.12
Semantic	97.6	0.21	90.6	2.17
Sneaky	97.65	89.02	89.47	82.1
Atropos	96.97	88.26	81.57	74.45
Byzantine				
Attack Type	**Mean Accuracy**	**Mean Difference from Benign**	**Mean Accuracy**	**Mean Difference from Benign**
Pair Flip	97.68	1.12	92.5	3.76
Sym Flip	97.7	1.12	91.22	5.07
Random Noise	97.55	1.25	88.22	8.04
Lie Attack	97.6	1.2	88.76	7.5
Min Max	97.62	1.18	88.34	7.92
Min Sum	97.65	1.15	88.57	7.69
Sneaky Random 1	97.57	1.23	89.88	6.38
Sneaky Random 2	97.37	1.43	88.82	7.44
Sneaky Random 3	97.72	1.08	89.37	6.89
Sneaky Random 4	97.95	0.85	88.8	7.46
Sneaky Random 5	97.42	1.38	89.68	6.58
Inverted Gradient	97.14	1.74	81.85	14.41
Gauss Images	97.2	1.6	87.27	8.99
Atropos	96.97	1.83	81.57	14.69

Table 2: 20% BCR DelphiFL V1 Data

7.3.1 Backdoor Attacks

Across the board, DelphiFL appeared to have a good handle on backdoor attacks.

Using MNIST, the attack success rate dropped all the way to 10.25%, which is much lower than its FedProx/FedAVG counterpart of 88.64%. It also successfully mitigated the percentage on all the other backdoor attacks. It decreased the attack success rate of semantic backdoor to 0.21% (from 1.58%). Sneaky backdoor did not decrease dramatically like base backdoor did, but it still decreased by quite a significant margin, going from 95.76% to 89.02%, which is close to a 7% decrease in the attack success rate from using the FedProx/FedAVG models. Atropos remained relatively unchanged.

In USPS, the same was observed, decreasing the attack success rates even more, notably for Atropos and sneaky backdoor. Where MNIST had close to a 7% decrease for sneaky backdoor, USPS had almost double that, with a 13.66% decrease in the attack success rate. This further shows how the USPS dataset is naturally more resistant to backdoor attacks. Atropos had a large decrease of 15%, dropping to 74.45% in terms of attack success rate. Base backdoor remained relatively unchanged compared to its FedProx/FedAVG USPS counterpart. Semantic backdoor also remained relatively unchanged compared to its FedProx/FedAVG USPS counterpart, but it was slightly more effective than the MNIST version of DelphiFL V1 (0.21% vs. 2.17%).

7.3.2 Byzantine Attacks

The byzantine attack discussion is a little different. Whereas DelphiFL V1 succeeded in mitigating backdoor attacks, byzantine attacks appeared to be more successful than backdoor attacks

With the MNIST dataset, DelphiFl V1 had all attacks in the range of 0.5% - 2%, whereas FedProx/FedAVG kept them in a 0.1% - 1.6% range. This is a 0.4% increase in the range. Even though there was an increase in the range, the values of each attack are still minimal. Some attacks that stood out to us were inverted gradient, gauss images, and Atropos. Table 2 describes how the mean difference of gauss images is a 1.6% increase from its 1.01% counterpart in FedProx/FedAVG. However, inverted gradient and Atropos both had 1.5% increase in their differences, with inverted gradient now having a 1.74% difference and Atropos having a 1.83% difference. With the USPS dataset, these attacks are much more detrimental.

Where MNIST had the attacks fall in the 0.5% - 2% range, USPS had the attacks fall in the 3.75% - 15% range, with most attacks falling in the 6.5% - 9% range, along with some outliers as seen in Table 2. These outliers are once again inverted gradient, gauss images, and Atropos. With gauss images, we see a 7.5%, going from 1.6% in MNIST, to 8.99% in USPS. Continuing on, inverted gradient saw a large increase from 1.74% to 14.41%, causing the attack to perform 8.3 times better. This applies to Atropos in the exact same setting, going from 1.83% to 14.69%, causing the attack to perform 8.03 times better while using the USPS dataset. Even though we concurred that the USPS dataset is naturally more susceptible to byzantine attacks, DelphiFL appeared not to mitigate it as much as other, more traditional algorithms, and caused the attacks to do even more damage than they did with FedProx/FedAVG.

DelphiFL V1 appeared to be able to mitigate backdoor attacks well while struggling to mitigate byzantine attacks. MNIST helped DelphiFL V1 to mitigate the byzantine attacks more effectively, whereas USPS appeared to provide the same support for backdoor attacks.

7.4 DelphiFL V2 – 20% BCR Results

DelphiFL V2 is the second version of DelphiFL, specifically designed to try and mitigate the backdoor attack of sneaky backdoor, and to make the model more robust.

Backdoor				
	MNIST		USPS	
Attack Type	Mean Accuracy	Mean Attack Success Rate	Mean Accuracy	Mean Attack Success Rate
Base	97.71	10.25	87.81	10.19
Semantic	97.73	0.28	89.53	3.54
Sneaky	97.57	88.39	88.37	81.08
Atropos	96.91	88.22	81.93	74.91
Byzantine				
Attack Type	Mean Accuracy	Mean Difference from Benign	Mean Accuracy	Mean Difference from Benign
Pair Flip	97.67	1.13	92.36	3.9
Sym Flip	97.67	1.13	91.57	4.69
Random Noise	97.57	1.23	87.44	8.82
Lie Attack	97.53	1.27	89.04	7.22
Min Max	97.57	1.23	89.68	6.58
Min Sum	97.62	1.18	89.89	6.37
Sneaky Random 1	97.47	1.33	89.96	6.3
Sneaky Random 2	97.46	1.34	88.82	7.44
Sneaky Random 3	97.66	1.13	91.22	5.04
Sneaky Random 4	97.86	0.93	91.33	4.93
Sneaky Random 5	97.6	1.19	88.82	7.44
Inverted Gradient	97.14	1.65	81.3	14.96
Gauss Images	97.41	1.41	86.99	9.27
Atropos	96.91	1.89	81.93	14.33

Table 3: 20% BCR DelphiFL V2 Data

7.4.1 Backdoor Attacks

In both MNIST and USPS, DelphiFL V2 appeared to not have an effect against these backdoor attacks, keeping the attack success rate relatively unchanged. In Table 3, sneaky backdoor's attack success rate is 88.39% and 81.08% for MNIST and USPS respectively. Comparing this to DelphiFL V1's values of 89.02% and 82.1%, it shows a fluctuation in the values of one increasing slightly and the other decreasing slightly depending on the dataset. However, because no value will ever be the same in an identical test, it is worthwhile that these can fluctuate, so we classified it as relatively unchanged. The other attacks followed the same pattern for both datasets, showing that DelphiFL V2 worked almost identically in mitigating backdoor attacks.

7.4.2 Byzantine Attacks

The same conclusion can be made for byzantine attacks in DelphiFL V2 as for backdoor attacks in DelphiFL V2. There were ever so slight changes in each value that made it difficult to tell if the model had improved or not. Table 3 demonstrates the values of inverted gradient are 1.65% and 14.96% for MNIST and USPS respectively. DelphiFl V1's values of this attack were 1.74% and 14.41%, having the same fluctuation as the backdoor attacks have. This applied to all the other byzantine attacks as well. This aids in drawing the conclusion that DelphiFL V2 works almost identically to the mitigation that DelphiFL V1 is doing to byzantine attacks.

DelphiFL V2 so far appears to have an almost identical output to mitigating these attacks as DelphiFL V1 has. As we go on with further results, we will see how that is not the case.

7.5 10% and 30% Results

Where the last results had been shown with an attacker ratio of 20% (2 attackers out of 10 clients), we also decided to experiment with a smaller number of attackers and a larger number of attackers. For this, we ran tests with 1 attacker from the 10 clients, and 3 attackers from the 10 clients.

7.5.1 10% Results

Backdoor				
	MNIST		USPS	
Attack Type	**Mean Accuracy**	**Mean Attack Success Rate**	**Mean Accuracy**	**Mean Attack Success Rate**
Base	98.61	58.72	94.59	15.49
Semantic	98.73	0.28	95.1	0.75
Sneaky	98.48	88.85	95.24	87.6
Atropos	98.6	89.53	94.77	87.52
Byzantine				
Attack Type	**Mean Accuracy**	**Mean Difference from Benign**	**Mean Accuracy**	**Mean Difference from Benign**
Pair Flip	98.72	0.08	95.1	1.16
Random Noise	98.32	0.48	93.63	2.63
Sneaky Random 2	98.83	-0.03	94.98	1.28
Inverted Gradient	98.63	0.17	95.3	0.96
Gauss Images	98.31	0.49	94.93	1.33
Atropos	98.6	0.2	94.77	1.49

Table 4: 10% BCR FedProx/FedAVG Data

FedProx/FedAVG

Moving over to having only 1 bad client in our tests, it is obvious that the attacks were not as successful as they were with the 2 bad clients. As seen in Table 4, there is a large difference in success rates in the Byzantine attacks.

Where with 2 bad clients for FedProx/FedAVG the byzantine attack accuracies ranged from 0.5% - 2% in MNIST, here it ranges from -0.05% - 0.5%, which is a large difference. Looking at sneaky random 4, the fact that the mean difference is negative shows that the attack is doing little damage to no damage at all. Not only that, but the backdoor attack success also went down, mostly in base backdoor. In Table 4, we can see how the mean attack success rate for base backdoor (using MNIST) is at 58.72%, instead of 88.64% as it was with 2 bad clients. This is a decrease of about 30%. Sneaky backdoor also had a 7% decrease, with semantic and Atropos not having large decreases, but still decreases nonetheless.

In USPS, we see the same trend. With 1 bad client, seen in Table 4, the percentage range for byzantine attacks is 0.9% - 2.7%, which is much lower than the original 2 bad client range (which was 1.2% - 11%). This decrease is large, especially given that USPS seems to be quite susceptible to byzantine attacks. Backdoor attacks are a little different. For semantic backdoor, there was about a 2% decrease, with 1 bad client causing the success rate to be 0.75% (whereas 2 bad clients caused the attack success rate to be 2.65%). However, looking at the other backdoor attacks, there was either little to no change, or the success rate was higher than the tests with 2 bad clients. We can see this in base backdoor, where the test with 2 bad clients had a base backdoor success rate of 11.44%, where the test with 1 bad client had a base backdoor success rate of 15.49%.

Backdoor				
	MNIST		USPS	
Attack Type	Mean Accuracy	Mean Attack Success Rate	Mean Accuracy	Mean Attack Success Rate
Base	97.7	10.23	90.45	9.94
Semantic	97.64	0.16	89.68	1.21
Sneaky	97.63	87.81	89.77	82.16
Atropos	97.49	88.64	87.08	80
Byzantine				
Attack Type	Mean Accuracy	Mean Difference from Benign	Mean Accuracy	Mean Difference from Benign
Pair Flip	97.62	1.18	90.54	5.72
Random Noise	97.6	1.2	92.02	4.24
Sneaky Random 2	97.78	1.02	89.77	6.49
Inverted Gradient	97.57	1.23	88.1	8.16
Gauss Images	97.61	1.19	91.32	4.94
Atropos	97.49	1.31	87.08	9.18

Table 5: 10% BCR DelphiFL V1 Data

DelphiFL V1

With DelphiFL V1, we can see that, overall, there does not seem to be a huge amount of change in both datasets, in both types of attacks. The ranges are basically the same (staying around 1% - 2% for MNIST, and 4% - 9% for USPS). Even so, the actual differences have slightly deceased. Where in the 2 bad clients tests, the percentage in MNIST was 1.83% and in USPS was 14.69% for Atropos, it fell off a bit in the 1 bad client tests, decreasing to 1.31% and 9.18% respectively. Backdoor attacks are also relatively unchanged, decreasing only marginally, if at all. In Table 5, we can see that the mean attack success rates have decreased slightly from the original values as shown in Table 2. This helps the idea that the success rates of attacks also depend on the percentage of attackers in the training phase.

Backdoor				
	MNIST		USPS	
Attack Type	Mean Accuracy	Mean Attack Success Rate	Mean Accuracy	Mean Attack Success Rate
Base	97.56	10.18	90.51	9.95
Semantic	97.63	0.48	88.7	1.9
Sneaky	97.61	87.82	89.38	81.74
Atropos	97.57	88.64	87.14	79.96
Byzantine				
Attack Type	Mean Accuracy	Mean Difference from Benign	Mean Accuracy	Mean Difference from Benign
Pair Flip	97.71	1.09	91.22	5.04
Random Noise	97.56	1.24	91.4	4.86
Sneaky Random 2	97.68	1.12	88.9	7.36
Inverted Gradient	97.51	1.29	88.2	8.06
Gauss Images	97.48	1.32	91.3	4.96
Atropos	97.57	1.23	87.14	9.12

Table 6: 10% BCR DelphiFL V2 Data

DelphiFL V2

DelphiFL V2 has the same results as DelphiFL V1, with the mean differences and mean attack success rates not showing much change in either dataset. We can see in Table 6 that the more detrimental attacks (Atropos and inverted gradient), decreased about the same amount as they did in DelphiFL V1. The only visible difference between V1 and V2 is that in V2 the base backdoor in USPS success rate does actually drop instead of rising as it did in V1.

7.5.2 30% Results

Backdoor				
	MNIST		USPS	
Attack Type	Mean Accuracy	Mean Attack Success Rate	Mean Accuracy	Mean Attack Success Rate
Base	98.57	96.06	91.52	11.83
Semantic	98.15	3.31	93.83	4.93
Sneaky	98.31	96.46	94.27	91.1
Atropos	98.14	89.2	93.12	86.63
Byzantine				
Attack Type	Mean Accuracy	Mean Difference from Benign	Mean Accuracy	Mean Difference from Benign
Pair Flip	98.46	0.34	94.44	1.82
Random Noise	96.13	2.67	80	16.26
Sneaky Random 2	98.57	0.23	94.48	1.78
Inverted Gradient	97.84	0.96	94.22	2.04
Gauss Images	72.49	26.31	92.2	4.06
Atropos	98.14	0.66	93.12	3.14

Table 7: 30% BCR FedProx/FedAVG Data

FedProx/FedAVG

When we test with 3 bad clients, you can see that there are quite some differences in the metrics, especially for byzantine attacks.

In MNIST, the most notable change is the mean difference for gauss images. In the normal 2 bad client tests, gauss images have a mean difference of 1.01% as seen in Table 1. In Table 7, we can see that the mean difference shoots up all the way to 26.31%. This is a large change, increasing the difference by 25%. We can also see that every other byzantine attack that was tested had an overall increase in the mean difference. Looking at backdoor attacks using MNIST, there is not anything outstanding other than an overall increase in the attack success rates, specifically in base and semantic backdoor. In Table 7, it shows that the mean attack success rate for base backdoor is 96.06% for the test with 3 bad clients, comparing that to the success rate of base backdoor for the test with 2 clients, which was 88.64%.

In USPS, the same case appears. In Table 7, we can see that the byzantine attacks appear to have increased by a small margin, besides one, increasing with a significantly larger margin. In Table 1, we see that (in USPS), the mean difference of random noise was 8.45%. In Table 7, we see that the mean difference of random noise jumps up to 16.26%, almost doubling the percentage with adding one more bad client. In backdoor attacks, USPS tests appear to have the same trend as MNIST tests in this preference, where the backdoor attack success rates have a small increase overall. In Table 7, the mean attack success rate for sneaky backdoor is up to 91.1%, which is an increase from the 2 bad client rate tests in Table 1, and the success rate for mean attack sneaky backdoor is 87.36%, which is about a 4% increase.

Backdoor				
	MNIST		USPS	
Attack Type	**Mean Accuracy**	**Mean Attack Success Rate**	**Mean Accuracy**	**Mean Attack Success Rate**
Base	97.56	10.5	89.99	11.92
Semantic	97.55	2.18	89.31	3.88
Sneaky	97.51	91.69	87.68	80.29
Atropos	96.09	87.65	69.78	62.46
Byzantine				
Attack Type	**Mean Accuracy**	**Mean Difference from Benign**	**Mean Accuracy**	**Mean Difference from Benign**
Pair Flip	97.62	1.18	90.23	6.03
Random Noise	97.32	1.48	88.76	7.5
Sneaky Random 2	97.41	1.39	87.38	8.88
Inverted Gradient	77.78	21.02	68.48	27.78
Gauss Images	96.35	2.45	82.57	13.69
Atropos	96.09	2.71	69.78	26.48

Table 8: 10% BCR FedProx/FedAVG Data

DelphiFL V1

With DelphiFL V1, looking at the overall change (comparing Table 8 to Table 2) with the MNIST dataset, there is a slight increase in all the byzantine attacks, as expected. However, there is one attack that did much better than the others: inverted gradient. In Table 8, we can see that the mean difference of the inverted gradient attack (using the MNIST dataset) is 21.02%, which is quite a large percentage. In Table 2, which is the 2 bad clients tests, we see that the mean difference of the inverted gradient attack is only 1.74%, meaning that this was 20% jump, which is quite large. Looking into backdoor attacks, it follows the same trend as the FedProx/FedAVG tests, where there is just a slight increase in all of the attacks, anywhere between a 1% - 4% increase.

Using the USPS dataset, the byzantine attacks have a much higher increase in their metric (mean difference from benign). In Table 8, we can see in the USPS section that the last 3 attacks have a huge increase in their mean difference. Inverted gradient increases to 27.78%, gauss images is at 13.69%, and Atropos has 26.48%. Comparing these to the values of Table 2 (14.41%, 8.99%, and 14.69% respectively), we can see that inverted gradient and Atropos both increased by about 13%, where gauss images increased by about 4%. Even though we know that USPS is more susceptible to byzantine attacks, it is still quite intriguing to see the amount of increase the metric had by adding only one more attacker. For the backdoor attacks, it appears that the attack success rate actually decreases rather than increases. Looking at Table 2, we can see that Atropos' attack success rate for USPS is 74.45%, whereas in Table 8, Atropos' attack success rate for USPS is 62.46%, which decreases it by about 13%. We can also see the same trend for the other backdoor attacks, not as prominent as Atropos, but still decreasing nevertheless.

Backdoor				
	MNIST		USPS	
Attack Type	**Mean Accuracy**	**Mean Attack Success Rate**	**Mean Accuracy**	**Mean Attack Success Rate**
Base	97.67	10.54	89.57	9.99
Semantic	97.42	3.21	88.93	3.54
Sneaky	97.48	89.68	88.5	81.27
Atropos	96.42	87.89	73.31	66.12
Byzantine				
Attack Type	**Mean Accuracy**	**Mean Difference from Benign**	**Mean Accuracy**	**Mean Difference from Benign**
Pair Flip	97.51	1.29	91.11	5.15
Random Noise	97.35	1.45	89.13	7.13
Sneaky Random 2	97.4	1.4	87.95	8.31
Inverted Gradient	95.55	3.25	69.85	26.41
Gauss Images	96.39	2.41	83.04	13.22
Atropos	96.42	2.38	73.31	22.95

Table 9: 10% BCR FedProx/FedAVG Data

DelphiFL V2

DephiFL V2 once again has a very similar trend to DelphiFL V1, with a few little differences.

In the MNIST dataset, the byzantine attacks have all increased slightly, which is a little different from DelphiFL V1, specifically with the inverted gradient attack. In Table 8, we see that the mean difference for inverted gradient is 21.02%, whereas in Table 9, the mean difference for inverted gradient is 3.25%. This shows that V2 handled the inverted gradient attack much better than V1 did. Comparing all the byzantine attack values to the ones in Table 3, we can see that there is definitely a slight increase on how well they manipulated the accuracies of the model. For backdoor attacks with the MNIST dataset, we can see that most of the attacks are relatively unchanged compared to the values in Table 3, except for semantic backdoor. Semantic backdoor with 3 bad clients had a mean attack success rate of 3.21%, as we can see in Table 9. This is about a 3% increase. This still is not an outstanding success rate, however.

In the USPS dataset, the byzantine attacks also follow the same pattern as DelphiFL V1. Once again, in Table 9, we see that the last 3 attacks have much higher mean differences compared to the other 3 byzantine attacks. This is shown with the inverted gradient attack having a 26.41%, gauss images having a 13.22%, and Atropos having a 22.95% difference. As we see in Table 3 with USPS, we see that the values for these 3 attacks respectively are: 14.96%, 9.27%, 14.33%. This means that the inverted gradient attack had 12% increase, gauss images had 4% increase, and Atropos had 9% increase. Now, looking at backdoors, we see a similar trend to DelphiFL V1. In Table 9, we can see that Atropos' attack success rate is 66.12%, which, compared to Table 3's value, is 8% decrease in the success rate. The other attacks however, have little to no differences from the values that are recorded in Table 3.

7.6 Analysis

7.6.1 DelphiFL

Version Similarities

Both versions of DelphiFL, as shown, appear to handle attacks relatively the same. As shown in most of the attacks (in both MNIST and USPS, with a 10%, 20%, or 30% bad client rate), DelphiFL V1 and V2 both have similar percentages in each of the metrics. For example, with the 20% bad client rate in MNIST, both DelphiFL V1 and DelphiFL V2 keep the range of 0.5% - 2% for each attack. Looking at the same bad client rate but at the USPS dataset, they also have the same range of 3% - 15%. This also follows the same pattern with the bad client rate being 10%, lowering each of the mean differences. We also see it with a 30% bad client rate, where the mean differences all increase, especially gauss images and Atropos (inverted gradient attack was the only case which we will touch on later). For backdoor attacks, we also see the same trend, with the attack success rates being similar. Atropos lowered to around 60% for the USPS dataset with a 30% bad client rate.

Benign Underperformance

We also see that when there are no attacks being run against both versions of DelphiFL, they appear to underperform compared to FedProx/FedAVG tests. As we know, using FedProx/FedAVG while using MNIST, the test had a mean accuracy of 98.8%. When we ran benign tests for DelphiFL V1 for MNIST with no attacks, we found that the mean accuracy was 97.57% and 97.73%. Running DelphiFL V2 for MNIST with no attacks, we found that the mean accuracy was 97.61% and 97.79%. As we can see, all of these values are at least 1% less than the FedProx/FedAVG benign mean accuracy.

V2 Robustness

One of our findings is that, using the MNIST dataset, DelphiFL V2 is more robust than DelphiFl V1. This became apparent due to the conclusions from the inverted gradient attack (when the bad client rate is 30%). For DelphiFL V1, we know that the mean difference of the inverted gradient attack was 21.02%, which is very high. However, when we ran it through the same conditions on DelphiFL V2, we found the mean difference of the inverted gradient attack dropped to 3.25%. This shows that, for MNIST, DelphiFL V2 is more robust and able to handle byzantine attacks a little better than DelphiFL V1.

Robustness in General

As we can see comparing both DelphiFL versions to the FedProx/FedAVG, DelphiFL is, overall, more robust to a broader range of attacks than FedProx/FedAVG. Looking at the preferences of a 20% bad client rate, even though FedProx/FedAVG seems to have an overall better handle on them, DelphiFL keeps them in better check. For example, with the base backdoor attack, FedProx/FedAVG has a 88.64% attack success rate in the MNIST dataset. With both DelphiFL versions, the attack success rate drops almost 80%, all the way down to 10% for both versions. Not only that, but if we look at the values of the 30% bad client rate tests, we see that gauss images for FedProx/FedAVG jump up to a 20% mean difference, whereas in DelphiFL it stays down to around 2% - 3%. This is especially true when we look at DelphiFL V2, where all of the byzantine attack mean differences stay around that 1% - 4%. This shows that DelphiFL is, overall, more robust to a broader range of attacks than FedAVG/FedProx.

7.6.2 Attacks

Sneaky Backdoor

One attack that we found to be very successful was the sneaky backdoor attack. As we have said, sneaky backdoor works by scrambling all the pixels in an image, causing the model to label the image as whatever the attacker chooses (2 in our case). In all of our tables, no matter the dataset or bad client rate, sneaky backdoor's attack success rate stayed above 80%. This was unlike any other backdoor attack that we tested. There were many cases where Atropos would drop to 60%, and even though base backdoor was very successful against FedProx/FedAVG, it was not as successful against the DelphiFL models, having the attack success rate drop all the way to 10%. This means that backdoor attacks have the opportunity to cause some serious damage if the task is ambiguous enough to not be noticed.

Atropos

Atropos was our first time attempting a combined attack. Even though there were many cases where it was not as successful (for example, dropping to around 60% for the 30% base client rate tests against DelphiFL with a USPS dataset), it was still doing relatively well compared to the other backdoor and byzantine attacks. Even though the attack success rate dropped to 60%, the mean difference skyrocketed to anywhere between 22% - 27%. Whereas with a 10% bad client rate, the mean accuracy was low, the mean attack success rate was high. This shows that in a combined attack with both backdoor and byzantine attacks implemented, when the bad client rate lowers, the backdoor type of attack has more success. When the bad client rate rises, the byzantine type of attack has more success.

Inverted Gradient Attack

The inverted gradient attack was an attack that we decided to experiment with. This attack was very interesting, as it was either very effective, or very ineffective. For example, against DelphiFL on the USPS dataset with a 20% bad client rate, the mean accuracy was all the way up to 14%. However, against FedProx/FedAVG with the same conditions, it failed to exceed even 2%. Looking at all the 10% bad client rate tests, inverted gradient attack performed rather poorly, and did not do much. However, when there was a 30% bad client rate, it performed exceptionally well, having a mean difference going all the way up to 22% (with the exception of FedProx/FedAVG, where it was still underperforming at around 2%). Looking at the MNIST dataset, it underperformed in almost every single test, besides against DelphiFL V1 with a 30% bad client rate, where it jumped up to 21%. This shows that not every attack that is implemented is going to be very successful against every single type of dataset and every model that is out there.

8. Future Work

Future work we would like to pursue would include: improving the simulation, adding a local counterpart to DelphiFL, incorporating network data, and developing additional attacks. Modifications to the simulation code itself include the following: Distributing the client simulations to run in parallel instead of linearly to improve speed of computation, further documenting the code (as the original repository had very little associated documentation), adding various quality of life improvements including error messages, exit codes, and flags, adding the capacity to run several attacks at once, including the ability to run on heterogeneous datasets, and finally, testing scalability of DelphiFLV1 and V2.

As demonstrated by our results, DelphiFlV1 and DelphiFLV2 are successful investigations into creating a more robust unified federation method; however, attacks which can make significant alterations to the functionality of the model, particularly in white box scenarios, can still be created. In particular, the Sneaky Backdoor attack we developed was exceptionally effective at inserting additional, undesired functionality into the model. As such, we investigated briefly if this particular attack could be mitigated by the addition of a local counterpart to DelphiFLV1 and V2. Initial findings were promising in detecting the attack through a means of anomaly detection via the utilization of statistical inference on the local scale. However, additional research in this direction is still needed.

At the beginning of our research, we decided to partner with Idaho National Labs and were given data (in the form of internet packets) from one of their softwares, Malcolm, to research. This data demonstrated what a typical 24 hour cycle of network activity looked like in a secure facility. We made progress, but ended up diverting most of our attention to DelphiFL. In the future, we would like to continue to research into the data by using anomaly detection algorithms. It is of great interest to our team to further the zero-trust aspect of DelphiV1 and V2 by utilizing packet capture data and anomaly detection software to examine and devalue in terms of model weight local models submitted from clients with suspicious information tied to their packet data (ie. requests from foreign actors). As such, further research into incorporating and developing packet capture data is a possible future direction.

We would also be very interested in continuing to create, analyze, and deploy increasingly sophisticated attacks to combat not only our own defenses, but also any other modern and future defense strategies. For example, in creating Atropos, we found that it is possible to create an attack combining other existing attacks in order to pursue multiple goals at once. Very few studies have been done into this, and even if such an attack is not as effective as its separate parts, given a malicious actor is not limited to one form of attack, it is still a worthy avenue of pursuit. Another potential area of research is that of a "sleeper agent" attacker. In a system utilizing zero-trust, one way to potentially bypass these security measures would be to initially provide benign models until a certain level of trust is reached, at which point the attacker would switch to submitting compromised models. Both of these would be useful in expanding the range of what a defense for FL could do.

Finally, we would like the opportunity to test the scalability of DelphiFLV1 and V2 as it is a multistep process including some methods that were argued to be unscalable. This, of course, would require the computational resources to run the simulation on many more clients at once, and would most likely need to occur after the aforementioned improvements to the simulation were made.

9. Conclusion

In summary, Blue Team created DelphiFL as an approach to improve defenses and test standard attacks and those created by Red Team. Blue Team was able to combine standard methods as a way to defend against two types of poisoning attack, Byzantine and Backdoor, whereas standard methods are only built to defend against certain types of attacks. FedAVG and FedProx are examples of these and were compared with DelphiFL. However, Red Team made two backdoor attacks that both standard methods and DelphiFL defended poorly against. Overall, both teams were successful in their research.

Conflicts of Interest

The authors declare no conflict of interest.

Author Contributions

Jonathan Flores: Conceptualization, Methodology, Software, Visualization, Investigation, Data Curation, Writing–Methods, Defenses, Future Work, References, Review & Editing. **Erin Kendall**: Software, Data Curation, Writing—-Abstract, Introduction, Attacks, Review & Editing. **Adam Crayton**: Software, Visualization, Investigation, Writing–Attacks, Results, Review & Editing. **Hailey Whipple**: Data Curation, Software, Writing–Background, Defenses, Future Work, Conclusion, Review & Editing.

Funding

This work is supported by funds from the National Science Foundation (NSF: # 2244596).

Data Availability

All code and data as well as additional resources related to this work can be found at: https://github.com/JonFlores3475/Summer2024REU/tree/main.

Acknowledgments

The authors would like to give special acknowledgement to Dr. Hao Chen and Zavareh Bozorgasl of the Boise State University Computer and Electrical Engineering and Computer Science Departments respectively for their direct supervision and guidance. The authors would also like to acknowledge the contributions to this work made by the Boise State University High Performance Computing and Research Computing teams via the allotment of access to the Borah Super Computer. Finally, the authors would like to acknowledge the contributions made by Dr. Fails and Dr. Yeh of the Boise State Computer Science Department as well as to the rest of the Boise State Computer Science Department for providing excellent guidance and the use of their facilities.

References

[1] Eugene Bagdasaryan et al. “How To Backdoor Federated Learning.” In: *CoRR* abs/1807.00459 (2018). arXiv: 1807.00459. URL: http://arxiv.org/abs/1807.00459.

[2] Arjun Nitin Bhagoji et al. “Analyzing Federated Learning through an Adversarial Lens.” In: *CoRR* abs/1811.12470 (2018). arXiv: 1811.12470. URL: http://arxiv.org/abs/1811.12470.

[3] Peva Blanchard et al. “Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent.” In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef Paper.pdf.

[4] Kallista Bonawitz et al. “Federated learning and privacy.” In: *Communications of the ACM* 65 (Apr. 2022), pp. 90–97. DOI: 10.1145/3500240.

[5] Di Cao et al. “Understanding Distributed Poisoning Attack in Federated Learning.” In: *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. 2019, pp. 233–239. DOI: 10.1109/ICPADS47876.2019.00042.

[6] Ka-Ho Chow & Ling Liu. “Perception Poisoning Attacks in Federated Learning.” In: Dec. 2021, pp. 146–155. DOI: 10.1109/TPSISA52974.2021.00017.

[7] Xiuwen Fang & Mang Ye. “Robust Federated Learning with Noisy and Heterogeneous Clients.” In: *CVPR*. 2022.

[8] Xiuwen Fang, Mang Ye, & Xiyuan Yang. “Robust heterogeneous federated learning under data corruption.” In: *ICCV*. 2023, pp. 5020–5030.

[9] Prajjwal Gupta et al. “A Novel Data Poisoning Attack in Federated Learning based on Inverted Loss Function.” In: *Computers Security* 130 (2023), p. 103270. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2023.103270. URL: https://www.sciencedirect.com/science/article/pii/S0167404823001803.

[10] Wenke Huang, Mang Ye, & Bo Du. “Learn from others and be yourself in heterogeneous federated learning.” In: *CVPR*. 2022.

[11] Wenke Huang et al. “A Federated Learning for Generalization, Robustness, Fairness: A Survey and Benchmark.” In: *arXiv* (2023).

[12] Wenke Huang et al. “Federated Graph Semantic and Structural Learning.” In: *IJCAI*. 2023.

[13] Wenke Huang et al. “Few-Shot Model Agnostic Federated Learning.” In: *ACMMM*. 2022, pp. 7309–7316.

[14] Wenke Huang et al. "Generalizable Heterogeneous Federated Cross-Correlation and Instance Similarity Learning." In: *TPAMI* (2023).

[15] Wenke Huang et al. "Rethinking Federated Learning with Domain Shift: A Prototype View." In: *CVPR*. 2023.

[16] Matthew Jagielski et al. "Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning." In: *CoRR* abs/1804.00308 (2018). arXiv: 1804.00308. URL: http://arxiv.org/abs/1804.00308.

[17] Lingjuan Lyu et al. "Privacy and Robustness in Federated Learning: Attacks and Defenses." In: *CoRR* abs/2012.06337 (2020). arXiv: 2012.06337. URL: https://arxiv.org/abs/2012.06337.

[18] Chuan Ma et al. *Trusted AI in Multi-agent Systems: An Overview of Privacy and Security for Distributed Learning*. 2023. arXiv: 2202.09027 [cs.DC]. URL: https://arxiv.org/abs/2202.09027.

[19] El Mahdi El Mhamdi, Rachid Guerraoui, & Sébastien Rouault. *The Hidden Vulnerability of Distributed Learning in Byzantium*. 2018. arXiv: 1802.07927 [stat.ML]. URL: https://arxiv.org/abs/1802.07927.

[20] Andrea Ponte et al. *SLIFER: Investigating Performance and Robustness of Malware Detection Pipelines*. 2024. arXiv: 2405.14478 [cs.CR]. URL: https://arxiv.org/abs/2405.14478.

[21] Xinyi Shang et al. *Federated Learning on Heterogeneous and Long-Tailed Data via Classifier Re-Training with Federated Features*. 2022. arXiv: 2204.13399 [cs.LG]. URL: https://arxiv.org/abs/2204.13399.

[22] Vale Tolpegin et al. "Data Poisoning Attacks Against Federated Learning Systems." In: (July 2020). DOI: 10.48550/ARXIV.2007.08432. arXiv: 2007.08432 [cs.LG].

[23] Marc Vucovich et al. *FedBayes: A Zero-Trust Federated Learning Aggregation to Defend Against Adversarial Attacks*. 2023. arXiv: 2312.04587 [cs.CR]. URL: https://arxiv.org/abs/2312.04587.

[24] Jianping Wu, Jiahe Jin, & Chunming Wu. "Challenges and Countermeasures of Federated Learning Data Poisoning Attack Situation Prediction." In: *Mathematics* 12.6 (2024). ISSN: 2227-7390. DOI: 10.3390/math12060901. URL: https://www.mdpi.com/2227-7390/12/6/901.

[25] Geming Xia et al. "Poisoning Attacks in Federated Learning: A Survey." In: *IEEE Access* 11 (2023), pp. 10708–10722. DOI: 10.1109/ACCESS.2023.3238823.

[26] Xiyuan Yang, Wenke Huang, & Mang Ye. "Dynamic Personalized Federated Learning with Adaptive Differential Privacy." In: *NeurIPS*. 2023.

[27] Mang Ye et al. "Heterogeneous Federated Learning: State-of-the-art and Research Challenges." In: *CSUR* (2023).

[28] Mang Ye et al. “Revisiting Federated Learning with Label Skew: An Over-Confidence Perspective.” In: *SCIS* (2024).

[29] Jiale Zhang et al. “Poisoning Attack in Federated Learning using Generative Adversarial Nets.” In: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2019, pp. 374–380. DOI: 10.1109/TrustCom/BigDataSE.2019.00057.

[30] Ying Zhao et al. “PDGAN: A Novel Poisoning Defense Method in Federated Learning Using Generative Adversarial Network.” In: *Algorithms and Architectures for Parallel Processing: 19th International Conference, ICA3PP 2019, Melbourne, VIC, Australia, December 9–11, 2019, Proceedings, Part I*. Melbourne, VIC, Australia: Springer-Verlag, 2019, pp. 595–609. ISBN: 978-3-030-38990-1. DOI: 10.1007/978-3-030-38991-8_39. URL: https://doi.org/10.1007/978-3-030-38991-8_39.