NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SECURING MACHINE LEARNING SUPPLY CHAINS

by

Joshua D. Strubel

September 2021

Thesis Advisor:	Joshua A. Kroll
Co-Advisor:	Marko Orescanin

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC, 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2021	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE SECURING MACHINE LEARNING SUPPLY CHAINS			**5. FUNDING NUMBERS**
6. AUTHOR(S) Joshua D. Strubel			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Recent cyber-attacks on supply chains such as the large-scale SolarWinds attack are gaining the attention of cybersecurity experts. Supply chain attacks are growing in frequency and are taking advantage of the trust that organizations put in the dependencies of their supply. The machine learning supply chain is incredibly vulnerable to this category of attack because of the large number of dependencies utilized. We demonstrate a weakness in a machine learning supply chain by attacking the model's parameters. We then demonstrate how an organization can implement secure checkpoints that generate integrity metadata and detect this class of attack before proceeding to the next phase in the supply chain.

14. SUBJECT TERMS machine learning, software development life cycle, cyber security, model integrity			**15. NUMBER OF PAGES** 71
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

SECURING MACHINE LEARNING SUPPLY CHAINS

Joshua D. Strubel
Civilian, CyberCorps: Scholarship for Service
BS, Bob Jones University, 2018

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2021

Approved by: Joshua A. Kroll
Advisor

Marko Orescanin
Co-Advisor

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Recent cyber-attacks on supply chains such as the large-scale SolarWinds attack are gaining the attention of cybersecurity experts. Supply chain attacks are growing in frequency and are taking advantage of the trust that organizations put in the dependencies of their supply. The machine learning supply chain is incredibly vulnerable to this category of attack because of the large number of dependencies utilized. We demonstrate a weakness in a machine learning supply chain by attacking the model's parameters. We then demonstrate how an organization can implement secure checkpoints that generate integrity metadata and detect this class of attack before proceeding to the next phase in the supply chain.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION....1
- A. PROBLEM....1
- B. CONTRIBUTION....2
- C. THESIS ORGANIZATION....3

II. UNDERSTANDING SUPPLY CHAINS....5
- A. CLASSIFICATION OF SUPPLY CHAINS....5
- B. DEFINING SOFTWARE SUPPLY CHAINS....8
- C. MACHINE LEARNING SUPPLY CHAIN....10
- D. CONCLUSION....13

III. SECURITY WITHIN THE SUPPLY CHAIN....15
- A. SECURITY TECHNIQUES....16
- B. MACHINE LEARNING SUPPLY CHAIN....19

IV. DEMONSTRATING SUPPLY CHAIN RISKS AND MITIGATIONS....23
- A. SETTING AND THREAT MODEL....23
- B. DEMONSTRATION ATTACK....26
- C. MAINTAINING MODEL INTEGRITY....29
- D. IMPLEMENTATION....31
- E. EXPERIMENTAL RESULTS....33

V. CONCLUSION....39
- A. CONTRIBUTION....39
- B. FUTURE RESEARCH....40

LIST OF REFERENCES....43

INITIAL DISTRIBUTION LIST....49

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1. High-level View of the Agile Software Development Life Cycle. Sources: [22], [23]. 6

Figure 2. Agile Implementation in Larger Projects, with Multiple Components. Sources: [22], [23], [26]....8

Figure 3. Software Supply Chain. Sources: [22], [23]. 10

Figure 4. An Abstract Machine Learning Product Life Cycle. Source: [27]. 11

Figure 5. An Abstract Machine Learning Product Life Cycle. Source: [27]. 12

Figure 6. An Abstract Machine Learning Product Life Cycle. Source: [27]. 12

Figure 7. CIA Triad. Source: [33]. 16

Figure 8. An Abstract Machine Learning Product Life Cycle with Security Controls and Dependencies. Source: [22], [23]. 18

Figure 9. Model's Attack Surface. Source: [27]. 24

Figure 10. Threat Model. Source: [27]. 26

Figure 11. Needed Checkpoint. Source: [27] 29

Figure 12. Our Checkpoint. Source: [27] 30

Figure 13. MNIST Dataset Examples. Source: [44]. 32

Figure 14. Model Supply Chain. Source: [27]. 33

Figure 15. Change in Accuracy versus Target Class....34

Figure 16. Dataset Composition by Image Class (percentage) 35

Figure 17. Probability of Misclassification versus. Target Class....35

Figure 18. Attack Performance Using Custom Loss versus Target Class....36

Figure 19. Detection Performance....37

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. Data Collection Dependencies 20

Table 2. Dependencies of the Training Phase 21

Table 3. Dependencies in the Testing Phase 21

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AWS	Amazon Web Services
FGSM	fast gradient sign method
MNIST	Modified National Institute of Standards and Technology
NIST	National Institute of Standards and Technology
SDLC	software development life cycle
CIA	confidentiality, integrity, availability
ML	machine learning
DNN	deep neural network
TCB	trusted computing base
SBOM	software bill of materials
RMF	risk management framework
DFAR	Defense Federal Acquisition Regulation
SBA	single bias attack
DOS	denial of service
NSA	National Security Agency
API	application programming interface

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Recent cyber-attack such as the large-scale SolarWinds attack are gaining the attention of cybersecurity experts [1]. The SolarWinds attack, which affected over 100 large organizations, is an example of a supply chain attack [1]. A supply chain attack involves an adversary manipulating the dependencies of a deliverable, for example by inserting malicious code or new malicious components [2]. Cybersecurity experts are seeing an incredible increase in frequency of supply chain attacks [3]. These attacks take advantage of an organization's trust in software, hardware, or vendor [4]. Because of its large and growing dependency on tools and third-party platforms and lack of sufficient industry security standards, machine learning supply chains are incredibly vulnerable to attack [5]. The ability to identify the architecture of the model [6], [7], poison training data [8], craft malicious input [9], and more recently altering the parameters of the model itself [10], [11], are all techniques an attacker can use to help compromise a machine learning supply chain.

The goal of this thesis is to demonstrate a mitigation technique to one type of machine learning supply chain attacks using standard cybersecurity principles instead of overly complicated machine learning solutions. Because of the large attack surface of supply chains [12], analyzing every component and dependency in a supply chain would be a tedious and large undertaking. Additionally, when attempting to secure the identified component or dependency, we can never be 100% confident that the code we depend on has not been modified. This inability to achieve 100% confidence in the code is an example of the Halting problem [13], [14]. Because of this lack of confidence, when implementing security within a supply chain, we need a design with possible failure in mind and the ability to identify and locate that failure. Because each component is an application-specific endeavor, this thesis focuses on methods of identifying and locating compromises within the machine learning supply chain before they spread to dependent components. We achieve this through security controls in strategic points of the machine learning supply chain. These security controls generate integrity metadata and act as gate keepers between

phases of the supply chains preventing the spread of compromises. A possible implementation of checkpoints is seen in Figure 1.

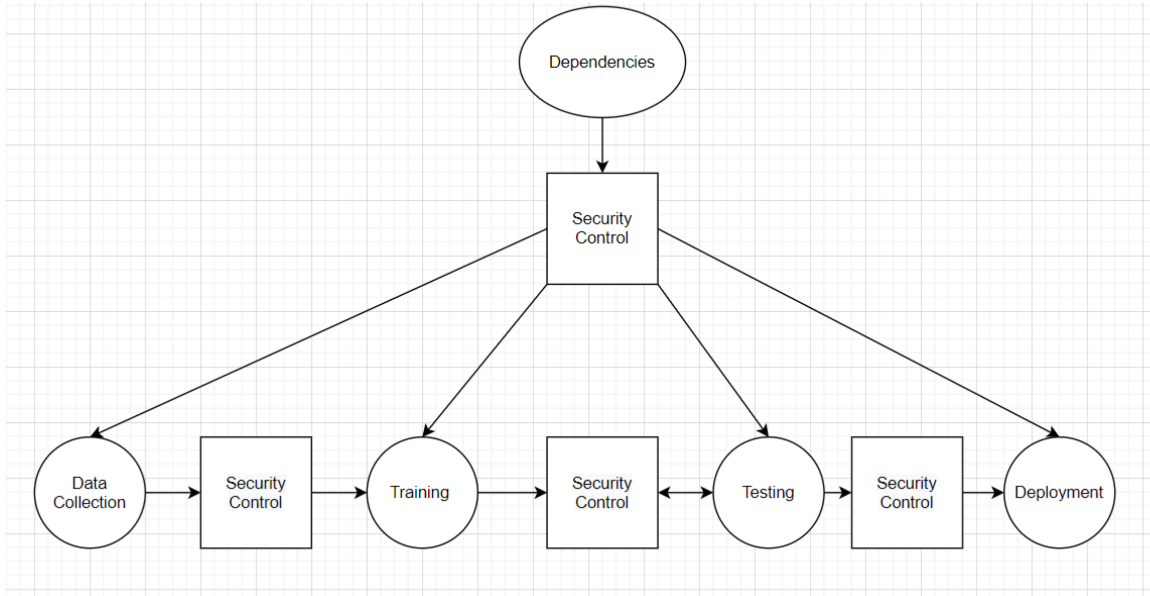

Figure 1. Static Machine Learning Product Life cycle with Security Controls. Source: [15]

To demonstrate the usefulness of this defense technique, we create our own machine learning application with its own supply chain. We simulate an attacker's ability to attack a machine learning model's integrity by accessing the saved location of our model and then perturbing the weights of the model.

After demonstrating the possibility of this attack and its potential damage, we design a security control. When constructing our security control, we wrap the standard TensorFlow checkpoint API calls to make a new custom checkpoint which generates and saves integrity metadata. This is simple and elegant. It can detect our attack and other attacks on model parameters. Because our checkpoint utilizes cryptographically secure hashes and hashes give fixed-size outputs and cryptographically secure hash functions have preimage and second preimage resistance, verification of the integrity metadata will fail in the same way if one parameter is changed or if every parameter of the model is changed

Our experiment demonstrates the vulnerability of our machine learning application to supply chain attacks. It also demonstrates how secure controls are a viable method of mitigating these attacks. Because our checkpoint utilizes hashes that show tampering regardless of the extent of supply-chain alterations, we are able to detect attack every time. This is not the only way to achieve integrity in a machine learning supply chain, but our technique is much simpler and cost effective than previously proposed machine learning techniques [16]-[19]. Future areas of research include methods to detect supply chain attacks on online machine learning applications and the usefulness of Merkle trees in locating the altered parameters.

References

[1] M. Willett, “Lessons of the solar winds hack,” *Glob. Polit. Strategy*, vol. 63, no. 2, pp. 7–26, Mar. 2021.

[2] A. Greenberg, “Hacker lexicon: What is a supply chain attack?,” *Wired*, May 2021, [Online]. Available: https://www.wired.com/story/hacker-lexicon-what-is-a-supply-chain-attack/

[3] “2020 state of the software supply chain.” Sonatype, 2020. Accessed: Aug. 10, 2020. [Online]. Available: https://www.sonatype.com/resources/white-paper-state-of-the-software-supply-chain-2020

[4] K. Thompson, “Reflections on trusting trust,” presented at the Turing Award Ceremony, San Francisco, CA, 1984.

[5] R. Kumar et al., “Adversarial machine learning—industry perspectives,” *SSRN Electron*. J., 2020, [Online]. Available: https://www.semanticscholar.org/paper/Adversarial-Machine-Learning-Industry-Perspectives-Kumar-Nystr%C3%B6m/3eb594bdc7057858a7bcd6243947c1944e89e2e3

[6] L. Liang et al., “DeepSniffer: A DNN model extraction framework based on learning Architectural Hints,” presented at the ASPLO’s 20, Lausanne, Switzerland, 2020.

[7] M. Yan, W. Fletcher, and J. Torrellas, “Cache telepathy: Leveraging shared resource attacks to learn DNN architectures,” presented at the 29th USENIX Security Symposium, San Diego, CA, Aug. 2020.

[8] M. Goldblum et al., “Dataset security for machine learning: data poisoning, backdoor attacks, and defenses,” *arXiv*, Dec. 2020, [Online]. Available: https://arxiv.org/abs/2012.10544

[9] I. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” presented at the ICLR, Mountain View, 2015.

[10] Y. Liu, L. Weu, B. Luo, and Q. Xu, “Fault injection attack on deep neural network - IEEE Conference Publication,” presented at the IEEE International Conference on Computer-Aided Design, Irvine, CA, Nov. 2017.

[11] J. Clements and Y. Lao, “Hardware trojan attacks on neural networks,” *arXiv*, Jun. 2018, [Online]. Available: https://arxiv.org/abs/1806.05768

[12] S. Eggers, “A novel approach for analyzing the nuclear supply chain cyber-attack surface,” *Nucl. Eng. Technol.*, vol. 53, no. 3, pp. 879–887, Sep. 2020.

[12] Alan Turing, “On computable numbers, with an application to the Entscheidungs Problem,” in Proceedings of the London Mathematical Society, vol. s2-42, pp. 230–265. Accessed: Aug. 03, 2021. [Online]. Available: https://academic.oup.com/plms/article/s2-42/1/230/1491926

[14] M. Harrison, W. Ruzzo, and J. Ullman, “On protection in operating systems,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 9, no. 5, pp. 14–24, Nov. 1976.

[15] A. Pelz-Sharpe and K. Kompella, “This is why you’re approaching your AI project wrong,” *Document Strategy*, Mar. 2019. https://documentmedia.com/article-2896-This-Is-Why-Youre-Approaching-Your-AI-Project-Wrong.html (accessed Aug. 25, 2021).

[16] Y. Liu, W. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, “ABS: scanning neural networks for back-doors by artificial brain stimulation,” in *ABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation*, 2019, pp. 1265–1282.

[17] X. Xu, Q. Wang, H. Li, N. Borisov, N. Gunter, and B. Li, “Detecting AI trojans using meta neural analysis,” *arXiv*, Oct. 2019, [Online]. Available: https://arxiv.org/abs/1910.03137

[18] K. Liu, B. Dolan-Gavitt, and S. Garg, “Fine-Pruning: defending against backdooring attacks on deep neural networks,” *arXiv*, May 2018, [Online]. Available: https://arxiv.org/abs/1805.12185

[19] M. Zou, Y. Shi, C. Wang, F. Li, W. Song, and Y. Wang, “PoTrojan: powerful eural-level trojan designs in deep learning models,” *arXiv*, Feb. 2018, [Online]. Available: https://arxiv.org/abs/1802.03043

ACKNOWLEDGMENTS

This work was conducted in partnership with the Naval Warfare Information Center through their Student Research Fellowship Program with the Naval Postgraduate School. Additionally, this material is based upon activities supported by the National Science Foundation under Agreement No 1565443. Any opinions, findings, and conclusions or recommendations expressed are those of the author and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

In late 2020, while the world was focused on the global Covid-19 pandemic, another pandemic, a cyber-pandemic, was discovered in America. SolarWinds, a software company based out of Texas, had been in business for over twenty years. They specialize in network monitoring and system management. Because of their great reputation, over one-hundred private companies and several federal agencies have relied on their services. Their client list included household names such as Microsoft, Intel, CISCO, the Department of Justice, and the Department of the Treasury. Like any software company, SolarWinds routinely updated software and sent security patches, and their trusting clients implemented these updates. As early as 2019, the Orion software developed by SolarWinds was infected by a sophisticated and well-funded hacker organization [1]. This hacker group embedded malware into one of the routine software patches sent out by SolarWinds [1]. The trusting clients implemented this software update and unknowingly were compromised. Because of the large number of SolarWinds' clients, the hacker group was able to infect an incredibly large number of organizations. This attack, commonly attributed to Russia, received an enormous amount of media coverage and caught the attention of the cyber-security community. The SolarWinds attack is interesting to the security community not necessarily because of the malware itself, but because of how it was delivered to the victims. The clients, many of whom had extensive security measures within their organization, were compromised not because of their own lack of security but because of their trust in software created outside their organization. The clients of SolarWinds were the victims of what is known as a supply chain attack [1].

A. PROBLEM

A supply chain attack is "a technique in which an adversary slips malicious code or even a malicious component into a trusted piece of software or hardware" [2]. The underlying vulnerability exploited by supply chain attacks is trust. When an organization utilizes tools, software, and hardware they have not created, they are implicitly trusting in the security of what is being used. Ken Thompson in his Turing Award Speech highlights

how trust can lead to vulnerabilities that are incredibly difficult to detect [3]. When an organization develops software, it must realize their place in the supply chain. The organization is part of a bigger process which includes vendors, tool developers, infrastructure support, and many other components. As the cyber-realm becomes more complex and dependent on third-party software and platforms, organizations are expanding the number of components in the chain and by extension their circle of trust and increasing their vulnerability to supply chain attacks.

Because of its large and growing dependency on tools and third-party platforms and lack of sufficient industry security standards [4], machine learning is incredibly vulnerable to supply chain attacks. Attackers have the ability to identify the architecture of the model [5], [6], poison training data [7], craft malicious input [8], and more recently, to alter the parameters of the model itself [9]–[11].

B. CONTRIBUTION

In this thesis, we build a basic machine learning application using standard tools and procedures. We then demonstrate its vulnerability to a supply chain attack. After demonstrating this vulnerability, we present a simple mitigation technique using standard cyber-security principles and procedures.

The principles of supply chains and their security, which we will identify in Chapters II and III, can be applied to all software development, but the scope of our experiment is limited to a subset of software development, machine learning. Within machine learning, the scope of our experiment is limited to static models trained on third-party platforms such as Colab or Amazon Web Services (AWS). Additionally, a major consideration in the selection of our experiment's methodology is simplicity. There are many complicated machine learning solutions that can address vulnerabilities in the machine learning supply chain [12]–[14], but we believe these complicated solutions are not entirely necessary. Any time complexity is added to a system, unexpected emergent behaviors are likely to occur [15]. The goal of our experiment is to demonstrate a possible mitigation technique to a specific machine learning supply chain attacks using standard cybersecurity principles instead of overly complicated machine learning solutions.

C. THESIS ORGANIZATION

The rest of this thesis is divided into four chapters. In Chapter II, “Understanding Supply Chains,” we take an in depth look at supply chains. In Chapter III, “Security Within the Supply Chain,” we will look at different security concerns within the supply chain. Chapter IV, “Demonstrating Supply Chain Risks and Mitigation,” details our demonstration of attacking and defending our machine learning application. We conclude our thesis and provide recommendations for future research in Chapter V, “Conclusion.”

THIS PAGE INTENTIONALLY LEFT BLANK

II. UNDERSTANDING SUPPLY CHAINS

Before we can defend supply chains, we need to define and understand them. We define a supply chain as anything that affects production of a given product. That definition is intentionally broad. The rest of Chapter II will be spent classifying supply chains, identifying components within a supply chain, and understanding the machine learning supply chain.

A. CLASSIFICATION OF SUPPLY CHAINS

There are two broad and non-mutually exclusive categories of supply chains: efficient and responsive [16], [17]. In industries where goods are easily substituted between producers, cost is the biggest differentiator for consumers, resulting in heavy emphasis on efficiency in their supply chain management. A good example of this would be the paper industry. When the average customer goes to the store to buy copier paper, they do not have an emotional attachment or even a preference for the International Paper company or Stora Enso company, they are primarily examining the cost difference between virtually indistinguishable products. We are not aware of any software developer that implements this type of supply chain, because of the general high specificity and sometimes limited substitutability of software [18].

The second broad category of supply chains is the responsive supply chain model. Whereas the focus of the efficient supply chain is efficiency, responsive supply chains are primarily concerned with their ability to quickly adapt to the changing needs of their clients. Industries more likely to use responsive supply chain models include the tech industries, marketing, and construction. Within this category are three non-mutually exclusive subcategories of supply chain models: flexible, custom-configured, and agile [19]. These categories and subcategories more closely resemble principles and primary goals than actual rules and specific procedures.

We focus here on agile supply chains, as they are often used in the software development industry [20]. This subcategory of supply chains does not start production before receiving customer specifications, so it is well suited to industries with

unpredictable demand. Software developers often achieve agility in their supply chains through the use of iterative development methodologies that derive and operationalize requirements during the development process [20]. Because it is of relevance to the development of modern machine-learning-based systems, we will only focus on the Agile software development methodology.

The concept of Agile software development was first popularized by the *Manifesto for Agile Software Development*, written by a group of developers in a resort located at Snowbird, Utah in 2001. The authors identified four principles for accomplishing this, valuing "individuals and interactions over processes and tools"; having "working software over comprehensive documentation"; encouraging "customer collaboration over contract negotiation"; and "responding to change over following a plan" [21]. Operationalizing these principles tends to force cyclical management of the supply chain. Figure 1 is a high-level representation of software developers' use of the Agile model.

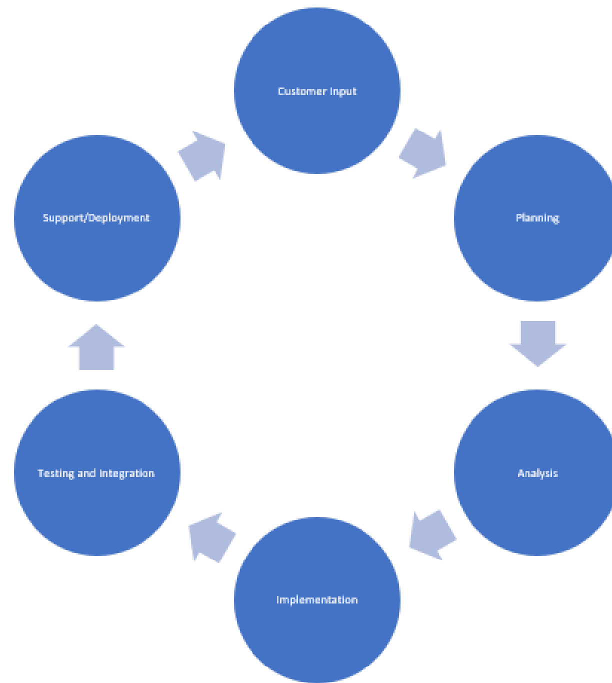

Figure 1. High-level View of the Agile Software Development Life Cycle. Sources: [22], [23].

Developers take input from customers throughout the development life cycle. Initial input is used to plan the project, provide early technical analysis of requirements, and reach a minimal viable product, while later input is used to build iteratively upon the existing product. In the implementation phase, coders turn the plans and analysis into code. As the product is developed, it is also tested for conformance to the customer's requirements. Testing can include functional testing, acceptance testing, or examining nonfunctional requirements [24], [25]. Testing and development may iterate until requirements are met satisfactorily. When the testing requirements are satisfied, the product enters an operational phase. During this phase, technical support of the application is either implemented or planned, which may reveal new requirements or revisions to existing requirements to be fed back into the beginning of the cycle. Cycles are meant to be completed quickly, to limit time wasted developing a product the customer is not happy with. On larger scale projects, separate components may be developed using the Agile method and later integrated into a final deliverable product, as shown in Figure 2.

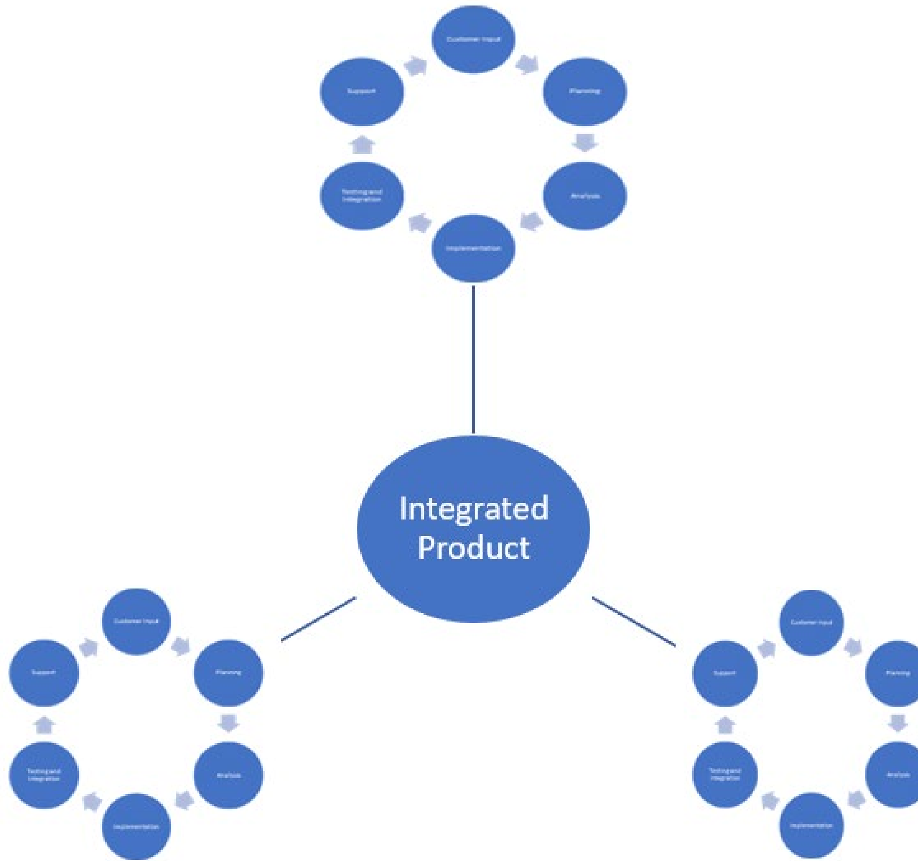

Figure 2. Agile Implementation in Larger Projects, with Multiple Components. Sources: [22], [23], [26].

B. DEFINING SOFTWARE SUPPLY CHAINS

Our discussion so far has dealt with principles of supply chains and the methodology of their management. However, even so informed, our understanding of supply chains for software is incomplete. We categorize software supply chains into four phases, outlined by the Agile model described above: development, testing of previously specified requirements, deployment, and support. These phases and their relationship to each other are shown in Figure 3. The development phase includes customer input, planning, analysis, and implementation. The testing phase is where testing and implementation of the product is conducted. Deployment is when the product is delivered to the customer. Information technology support, software patches, etc., happen in the support phase.

In addition to categorizing phases of the software supply chain for clarity, we consider the dependencies of each phase. We define dependencies of each life cycle phase to include any software, hardware, and personnel needed to accomplish the task associated with one of the phases. Dependencies under consideration vary from phase to phase and from product to product. Dependencies of every phase of the life cycle comprise the dependencies of the product overall. Any software that is used in the process would be a dependency. This would include the operating systems of the computers being used by the software developers, the assembler and complier used by the programmers, the platform the application is deployed to, and the tools used for testing the software. Hardware that could be considered a dependency would be the computers used by developers or onto which the software is deployed, and any physical infrastructure related to the project (datacenter racks, power distribution equipment, and computer networks). Personnel dependencies include employees of the developer as well as employees of its vendors and outside contractors. This includes employees involved during deployment, including those who provide technical support. Figure 3 highlights the role of dependencies in a software supply chain.

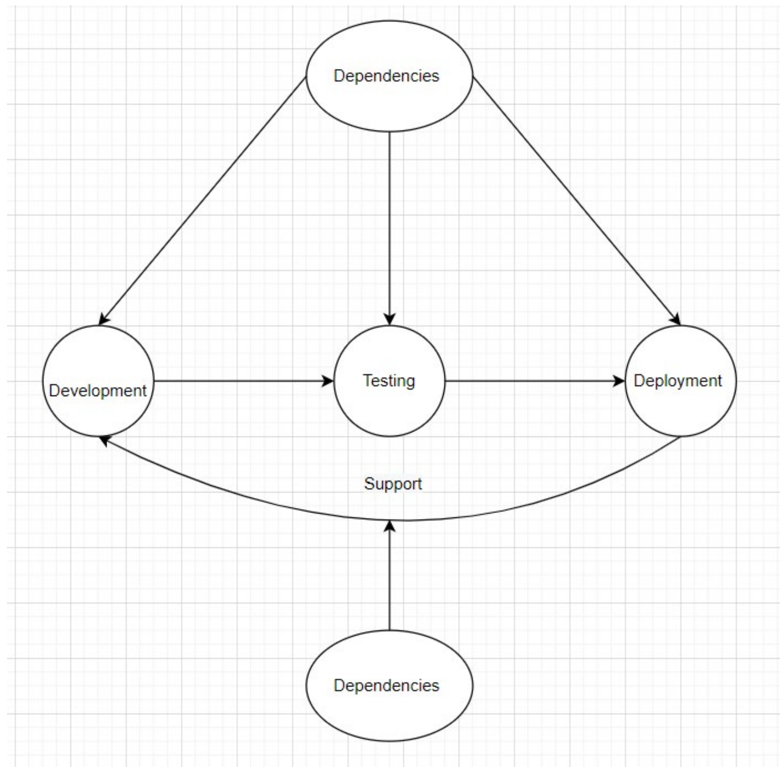

Figure 3. Software Supply Chain. Sources: [22], [23].

A deeper dive reveals that each of the identified dependencies has its own supply chain and dependencies. A change in the dependencies of dependencies of a product's supply chain may directly or indirectly affect the supply chain itself. To be precise: a supply chain is dependent on the closure of its dependences. This relationship between a supply chain and its dependencies underscores the question of trust in software development. By taking on a dependency within a software product's supply chain, we are trusting that dependency (and the closure of its dependencies). An additional consideration worth noting is the cyclical nature of software's supply chain management with regard to dependencies: a change in a dependency in one phase will eventually affect the rest of the supply chain.

C. MACHINE LEARNING SUPPLY CHAIN

There is not an accepted standard for the life cycle of a machine learning product. Figure 4 is a generic abstraction of a machine learning product life cycle.

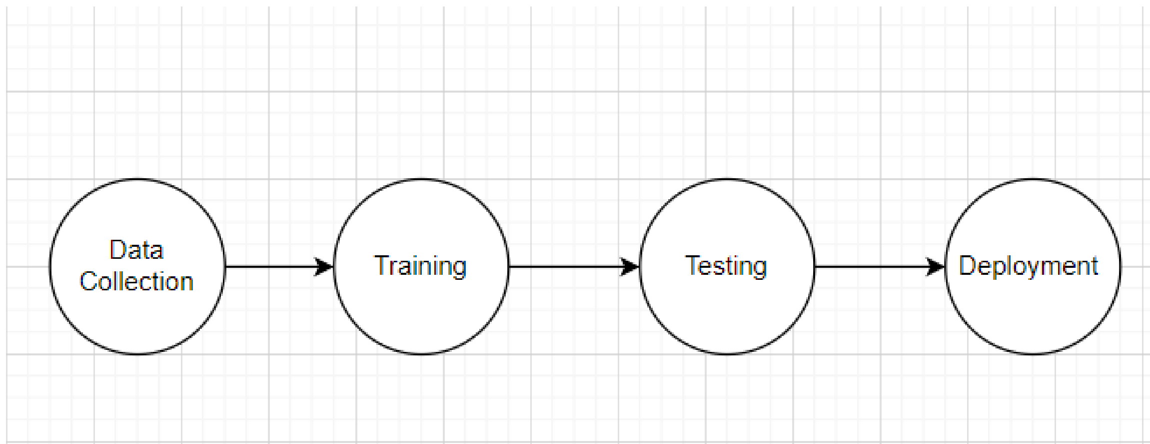

Figure 4. An Abstract Machine Learning Product Life Cycle. Source: [27].

The first phase in developing a machine learning product is data collection. Data collection includes acquiring, cleaning, and labeling data. Data collected will vary depending upon the specific application. For example, a machine learning application designed for distinguishing handwritten numerals would often require large numbers of images with handwritten numerals and the necessary formatting of data to be compatible with the input layer of the model. Data collected are often split into training data, validation data, and testing data. Training and validation data are used during the training phase, and testing data is used during the testing phase. Data collection is a very important phase because the quantity and quality of data has a significant impact on the performance of the model.

In the training phase, the model is trained using the training and validation datasets from the data collection phase; and in the testing phase, the model is tested on the testing dataset which is often intended to be disjoint from the training data.

The workflow described above is linear because it is not an online model. As stated previously, our experiment does not involve online models, so we mention them briefly only for completeness. In Chapter IV, we expound upon the importance of this distinction. An online model depends on a continuous supply of new data, feeding back data seen during the deployment phase as newly collected data. In an online model, developers train an initial model similarly to a traditional model. But upon deployment, the parameters of the model continuously change with the incoming feedback from new data. The life cycle of an online model resembles Figure 5.

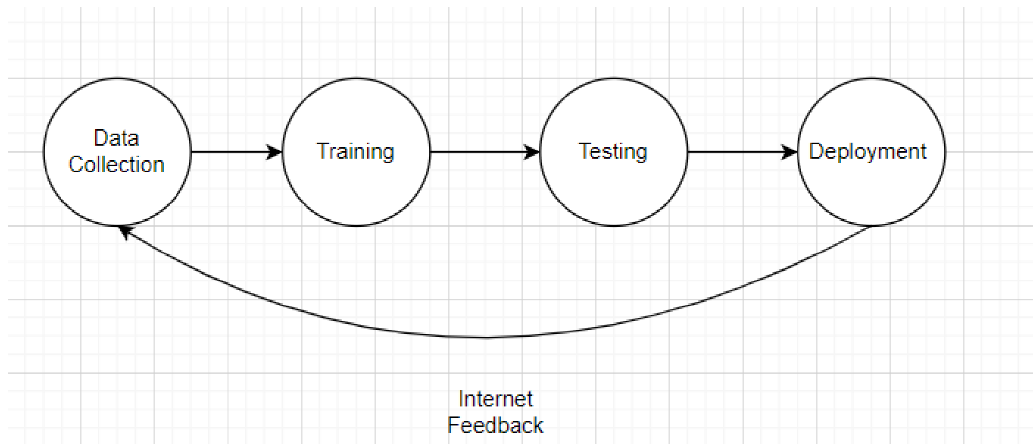

Figure 5. An Abstract Machine Learning Product Life Cycle. Source: [27].

As mentioned previously, a representation of a supply chain that does not include dependencies is not complete. A more fully developed representation of the product life cycle of a traditional machine learning application is shown in Figure 6.

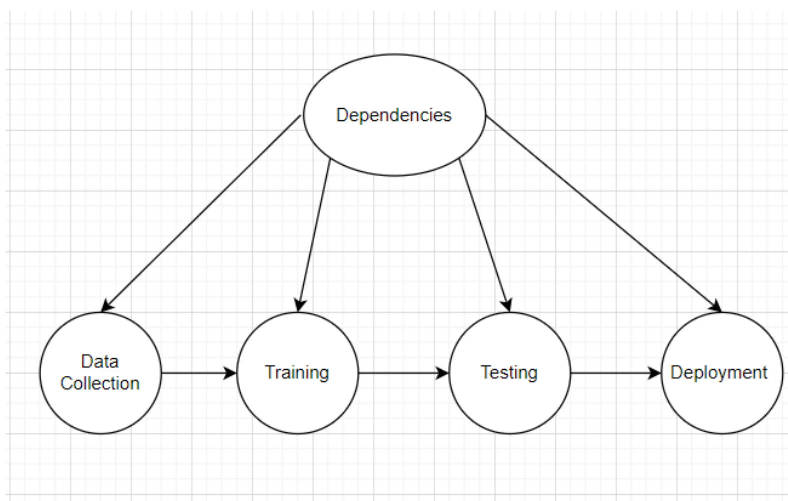

Figure 6. An Abstract Machine Learning Product Life Cycle. Source: [27].

Specific dependencies vary from application to application, but we highlight the most common, focusing on our chosen setting of developing and deploying a machine learning model on a third-party platform. In data collection, there are several dependencies: the provider of the data, labeling of data, cleaning of data, and the storage and retrieval of

the data. Providers of data can be the developers themselves, organizations such as National Institute of Standards and Technology (NIST) database, or companies that sell data to developers. Often the labeling and cleaning of data is also outsourced to different organizations. How data is retrieved from the provider and then stored is also a dependency that affects the data collection phase. In the next chapter, we examine how these dependencies can be exploited by malicious actors.

In the training phase, the model is trained using the training and validation datasets from the data collection phase; and in the testing phase, the model is tested on the testing dataset, which is often intended to be disjoint from the training data. Dependencies within the training phase and testing include but are not limited to the software and hardware being used to train the model, the platform the model is being trained on, employees or contractors working on the project, and tools and data used to test the model. Training machine learning applications is computationally expensive, leading many developers to train, test, and deploy machine learning models on third-party platforms such as AWS.

In the deployment phase, dependencies include the platform hosting the deployed product, which may or may not be the same platform as the development platform. Additional dependencies would include hardware such as the physical server storing the model [6].

D. CONCLUSION

In this chapter, we have laid the groundwork for a discussion of security within the supply chain, specifically machine learning supply chains. In Chapter III, we will examine the security implications of our discussion of supply chains. Before moving to next chapter, it is important to remember the key points of this chapter: a supply chain is dependent upon the closure of its dependencies, the cyclical nature of software supply chain management will propagate changes from one phase to the rest of the supply chain, and the machine learning supply chain has an abundance of dependencies.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SECURITY WITHIN THE SUPPLY CHAIN

Before we can discuss security within the supply chain, we need a high-level understanding of relevant cyber-security concepts. Traditionally, cyber-security discussions have revolved around the "CIA Triad" [28]. The CIA Triad is a common representation of three important concerns within cybersecurity: confidentiality, integrity, and availability. Confidentiality, "a concept that applies to data that must be held in confidence and that describes the status and degree of protection that must be provided for such data about individuals as well as organizations" [29], was popularized as a concern in Hofferberts' study for the Air Force in 1976 [30]. Discussions of confidentiality include topics such as encryption and access control. Discussions of integrity, the concept "which ensures that computer resources operate correctly and that data in the databases is correct" [29], include topics such as hashes, user verification, and checkpoints. Some of the early contributors to the idea of integrity within cybersecurity were Clark and Wilson at the 1987 IEEE Symposium on Research in Security and Privacy [31], and Biba in 1977 [32]. It is difficult to pinpoint the idea of availability to a single author. The cybersecurity dictionary *Data and Computer Security* defines availability as "the characteristic that ensures the computer resources will be available to authorized users when they need them" [29]. Availability is primarily concerned with failure of the system. Confidentiality, integrity, and availability are all connected concepts within the CIA Triad. The common representation of the relationship between these concepts is shown in the depiction of the CIA triad in Figure 7.

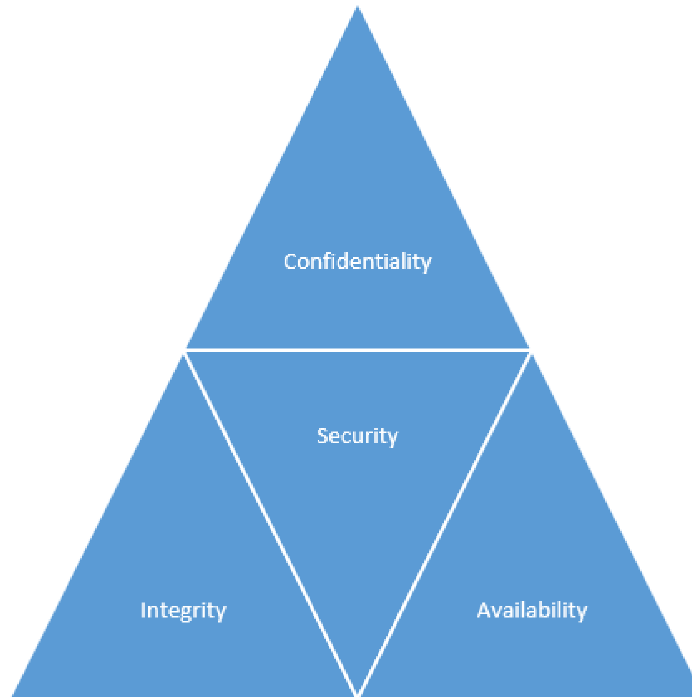

Figure 7. CIA Triad. Source: [33].

Supply chain attacks can attack the confidentiality, integrity, or availability of a supply chain. As we will see later, preventing the spread of compromises within the supply chain primarily deals primarily with integrity. An additional cybersecurity concept related to our discussion is a trusted computing base (TCB), which is defined as "the totality of protection mechanisms within a computer system—including hardware, firmware and software" [34].

A. SECURITY TECHNIQUES

In the previous chapter, we discussed how a change in any component of the supply chain influences the rest of the supply chain. A secure supply chain requires security measures at every stage of the supply chain [35]. Because of the number of components and the number of dependencies in a supply chain, the potential attack surface is large [36]. The first step in securing the supply chain is to identify the attack surface. This can be done be identifying the various components and dependencies of the application. A quick look at even a simple supply chain will reveal many components and dependencies. In Figure

6, we identified four phases and their dependencies in the abstracted machine learning supply chain. Common components and dependencies within these four categories include operating systems in use, software in use, contractors, vendors, platforms, and many other items.

Analyzing every one of these component and dependency, as in a Software Bill of Materials (SBOM), would be a tedious and large undertaking. Additionally, attempting to completely secure every identified component or dependency, as is the approach of the National Security Agencies' (NSA) Trusted Computer System Evaluation Criteria (TSEC) [37], does not provide complete confidence in the integrity of the analyzed code. This inability to achieve 100% confidence in the code is an example of the Halting problem [38], [39]. We can never be 100% sure that every component or dependency is perfectly secure. Because of this inability, when implementing security within a supply chain, we need a design with possible failure in mind and the ability to identify and locate that failure. Because of the application specificity of securing each component, the rest of this chapter will focus on methods of identifying and locating compromises within the supply chain before proceeding to the next phase.

One way to locate compromises within a supply chain is to add boundaries between phases within the supply chain. A visual representation of this can be seen in Figure 8. By strategically implementing security control, we can isolate the vulnerability. Possible security controls include items such as vendor questionnaires, code reviews, checksums, and cryptographic hash functions. These techniques are useful to our discussion, so we discuss them here.

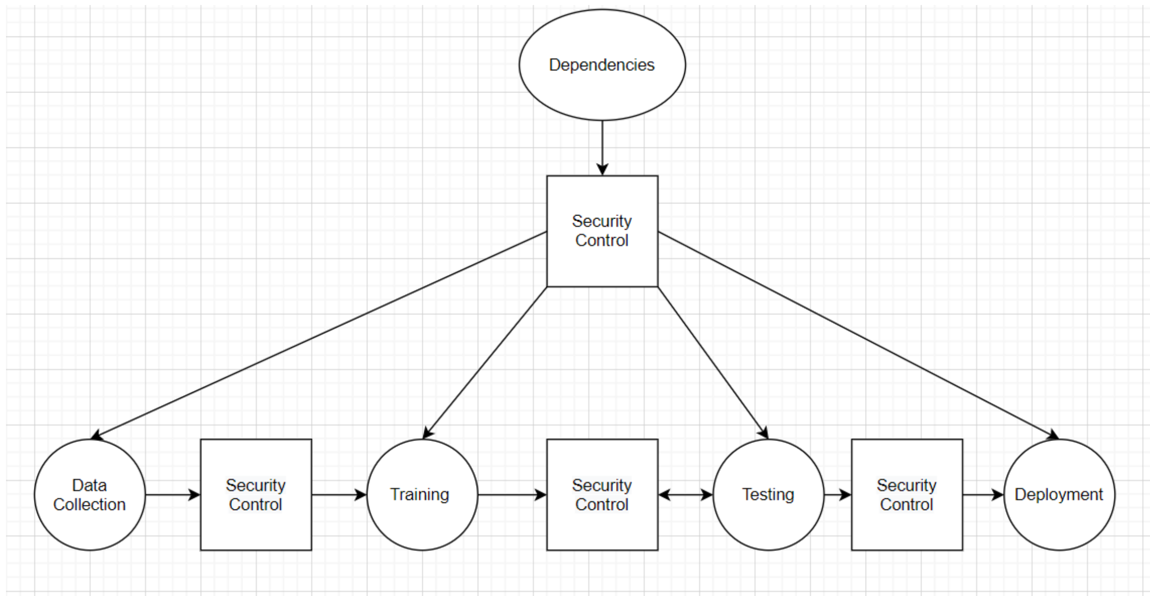

Figure 8. An Abstract Machine Learning Product Life Cycle with Security Controls and Dependencies. Source: [22], [23].

Vendor questionnaires are simple high-level security controls. Before using software from a vendor, a good practice is to require a vendor fill out a questionnaire detailing the specifics of the program. What questions are included in the questionnaire is up to the individual organization, but examples could include supported operating systems, delivery method of updates, libraries being used, etc. The government has less flexibility in these crafting questionnaires and the questionnaires are governed by various government standards such as the Risk Management Framework (RMF) from the National Institute of Standards and Technology (NIST). Upon completion, the organization decides if the risk of trusting the software outweighs the benefits of the software. While vendor questionnaires are good practice, they have underlying limitations [40].

Even the most well-intentioned vendor is sometimes hesitant to give detailed information on proprietary software. However, the government does get the source code for any software it orders from the developer for projects it solely funds under the guidelines of the Defense Federal Acquisition Regulation (DFAR), but possession of the source code in of itself does not ensure complete security of the code. An additional

security measure is third-party code reviews, which can be used as an alternative and or supplement to vendor questionnaires.

Vendors or customers occasionally hire third-party organizations to do code reviews of their products. Third-party organizations should not have a conflict of interest and so should present an impartial judgment on the product. Third parties as a security control are not foolproof, and they are expensive, but they are useful [40]. An additional layer of protection would be for the organization itself to examine the software before implementing it into their supply chain. Once again, this technique is not foolproof or without limitations. Many software programs are not open source, and it is expensive to conduct an extensive code review.

So far, the techniques discussed are high-level techniques. On the technical level, there are several techniques for verifying communication, users, and software identity (see, for example, approaches such as those in [41]–[43]). A simple technique that can be helpful are hashes. A hash is a function from an input domain, such as bytes, to a fixed-sized identifiers. In layman's terms, a hash is like a "fingerprint" for software. Any change in software will result in a different hash, regardless of the size of the change. Because hashes are statistically impervious to the size of the change, they can be very useful as a checkpoint when verifying that the software being used is the actual software provided by the already vetted vendor.

While none of these techniques is foolproof, adding secure checkpoints at strategic points in the supply chain will significantly increase chances of detecting compromises before proceeding to the rest of the supply chain. Up to this point, our discussion has been purposefully general. The goal of this thesis is to expound upon the security of machine learning supply chains. The rest of this chapter will look at ways to implement these strategic and secure checkpoints within a machine learning supply chain.

B. MACHINE LEARNING SUPPLY CHAIN

As discussed in Chapter II, there are four phases within the static machine learning workflow: data collection, training, testing, and deployment. The first step in hardening the supply chain is to identify the possible dependencies of a deliverable. We enumerate the

most common possibilities for each development phase. In Table 1, we list some of the common dependencies for data collection.

Table 1. Data Collection Dependencies

Hardware	Software	Personnel	Infrastructure
Data Storage	Software tools	Organization responsible for collecting, cleaning, and labeling data	Data transfer
	Operating Systems implementing software tools		Data governance systems and processes

Data collection is very important to the performance of the model. Ensuring the integrity and confidentiality of data is perhaps the most important ongoing activity. Malicious actors with access to data have the ability to poison the data, creating a compromised model [7]. A simple control point would involve examining the reputation of the outside parties, if any, responsible for collection, labeling, and cleaning of the data. Additionally, encrypting data could provide some added protection against confidentiality attacks.

Once data is securely collected, a security control, between the data collection phase and the training phase is needed. This security control should verify the data collected has not been altered, before continuing to the training phase. This added check helps mitigate the risk of potential data poisoning.

Within the training phase, are several dependencies. Training of machine learning models can be very computationally expensive. Because of this, many developers utilize third-party platforms such as AWS. It is important to realize that by using a third-party platform an organization is trusting the security of that third party. We want to know that the model being stored and trained on the platform has not changed over time or over space. Some other relevant dependencies are listed in Table 2.

Table 2. Dependencies of the Training Phase

Hardware	Software	Personnel	Infrastructure
Training data storage and computation	Machine learning support software (e.g., Tensor Flow)	Employees/contractors directly or indirectly involved in training the model	Third party platforms such as AWS
	Operating Systems		

Testing of a machine learning model often involves the same platform, tools, and software as the training phase, so the dependencies do not vary much; however, training and testing of a model are often done in many iterations (epochs). Because of these iterations, it is important before moving into the testing phase to have another security control as an extra layer of protection. At a minimum this control point should verify that the model at the end of the training phases is the same model being used at the beginning of the testing phase. Possible testing dependencies are seen in Table 3.

Table 3. Dependencies in the Testing Phase

Hardware	Software	Personnel	Infrastructure
Testing data storage and computation	Machine learning support software (e.g., TensorFlow)	Employees/contractors testing the model	Third party platforms such as AWS.
	Operating Systems		

Before transitioning to the deployment phase, we need another security control to verify that the model has not been tampered with. Once past this security control, the security implications really depend on the specific application, but adversarial examples are a common security concern. Deployment dependencies again are primarily concerned with the cloud platform.

Security control points are not new or complicated concepts in cybersecurity, but they can be very useful in catching breaches in security before they have a chance to spread to the rest of the supply chain. As mentioned previously, current industry best practice does not ensure comprehensive security of the software or machine learning supply chain [4]. Implementing security controls should be implemented into industry standards. Chapter IV describes a demonstration showing that machine learning supply chains without sufficient security controls are vulnerable to tampering. Additionally, Chapter IV presents techniques, which effectively mitigate this type of tampering.

IV. DEMONSTRATING SUPPLY CHAIN RISKS AND MITIGATIONS

We provide an experimental demonstration of one of the vulnerabilities of a machine learning application to a supply chain attack, in order to validate the risks described in Chapter III. We also demonstrate how to mitigate a class of supply chain attacks that include our demonstration attack. Our goal is not to create a new type of supply chain attack or a new defense technique, but to validate knowledge of the risks and controls in an exemplary setting analogous to commonly encountered supply-chain conditions. We show how implementing simple time-tested cybersecurity techniques in the form of secure ML training checkpoints can detect this specific attack. For clarity of this demonstration, we intentionally keep the machine learning application, the supply chain attack, and the secure checkpoint simple.

A. SETTING AND THREAT MODEL

In this section, we will first discuss the attack surface of our model, the selected threat model, and then mechanics of the attack and defense. As previously discussed, the attack surface of a supply chain can be very large [36]. The following discussion is not an exhaustive list of possible attacks on our supply chain, but a highlight of the most common can be seen in Figure 9. When crafting an attack on the supply chain, we can poison the dataset [7]. This would involve inserting several entries in the dataset with the wrong labels. For example, if we wanted to cause our model to misclassify all images of the numeral three, we could insert multiple images of the numeral three with a false label of another numeral into the training dataset. This could cause the model to misclassify the numeral three upon deployment.

Another option would be to attack the model after it has been deployed. We could carefully create inputs known as adversarial examples to cause the model to misclassify an image. The most well-known input attack is the Fast Gradient Sign Method (FGSM). In this attack, the attacker creates an adversarial example by taking a gradient of the loss of the model with respect to the input image and adding these perturbations to the image [8].

A famous example of this would be adding perturbations to the pixels in a stop sign image causing the model to misclassify the image.

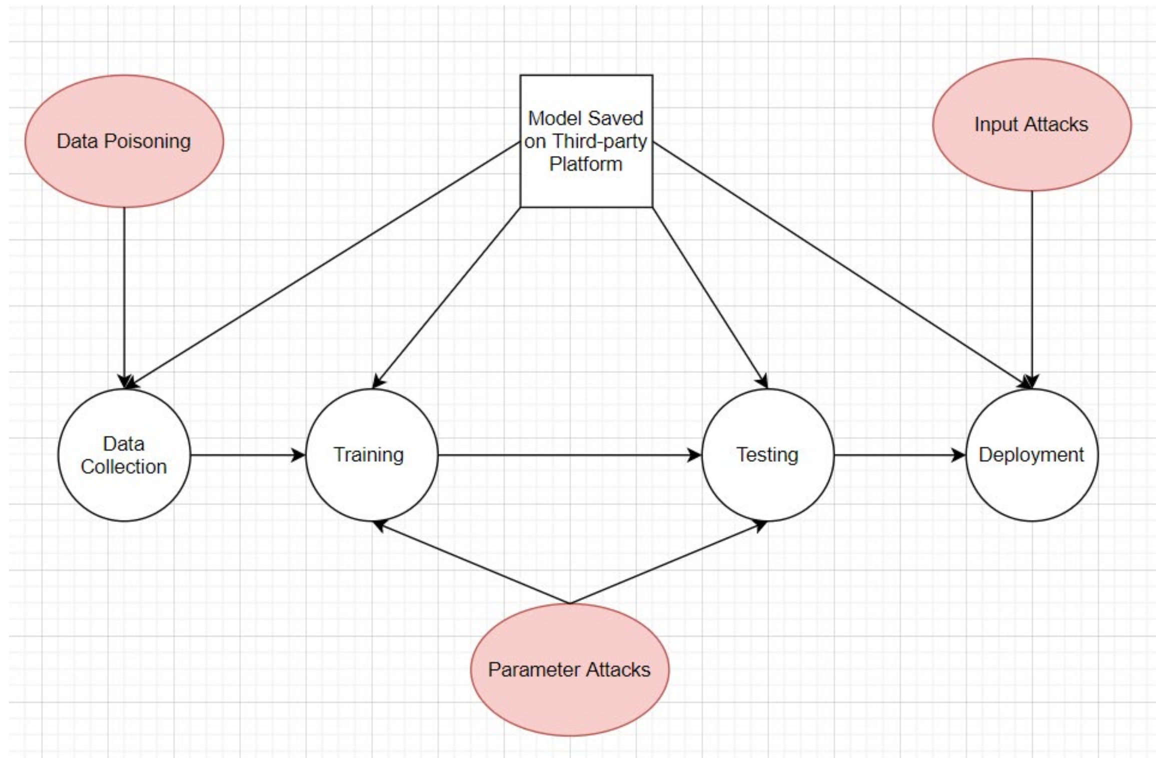

Figure 9. Model's Attack Surface. Source: [27].

Both attacks are viable ways of attacking the machine learning supply chain, but there are many research papers on defending these types of attacks. Because we want to highlight machine learning's dependency on third-party platforms, we chose to attack the integrity of the model by attacking the parameters. We will define and discuss our attack on parameters; but first we will define our threat model.

In our threat model, our attacker is someone with white-box access, whose goal is to change the output of a machine learning classification system trained and stored on a third-party platform. The attacker has knowledge of the system's architecture, access to the model, and the ability to modify the parameters of the model. We also assume the organization has access to secure storage outside of the third-party platform and that the attacker does not have access to this secure storage.

Let us examine our assumptions and investigate their reasonableness. Our first assumption is that the attacker has white-box access to our machine learning system. This access would require the credentials needed to access the third-party platform. The ability of a malicious actor to steal credentials has been demonstrated countless times and is a reasonable assumption [45]. The second assumption is that the attacker has knowledge of the system's architecture, access to the model, and the ability to modify parameters. Our previous assumption involved acquiring credentials to the third-party platform. If the attacker has these credentials, it is a reasonable assumption that they can view, access, and alter the machine learning model with the stolen credentials. We also assume the organization has access to secure storage outside of the third-party platform and that the attacker does not have access to this secure storage. This is an unorthodox assumption, but we believe it is reasonable. Compromising one set of credentials, does not automatically imply access on other systems.

We have described our threat model. The question remains, does this threat model resemble any real-life attacks? The answer is yes. Here is a real-life scenario where this threat model would be applicable. An organization trains and host a machine learning application on a third-party platform. An attacker steals the credentials of an employee with access to the third-party platform through social engineering, brute-force attack, or some other means. The attacker uses these stolen credentials to log into the third-party platform, access, view, and alter the model. The attacker tries to access the file systems outside of the third party that may hold metadata of the model's architecture, but access to these files requires credentials they do not hold. Each one of these actions is feasible, reasonable, and fits within our threat model. A representation of our threat model is shown in Figure 10.

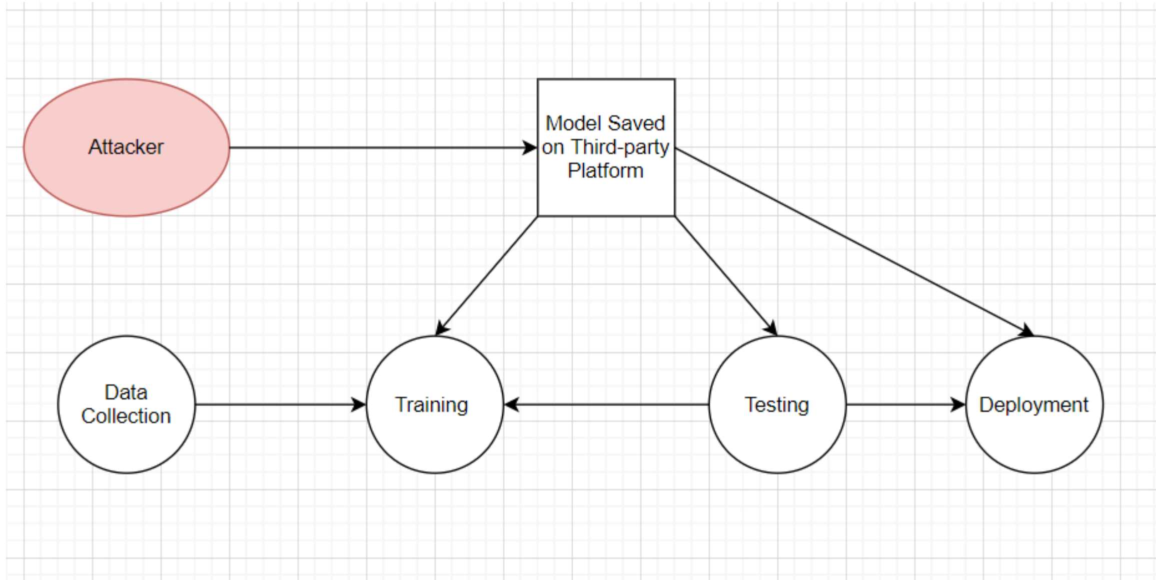

Figure 10. Threat Model. Source: [27].

B. DEMONSTRATION ATTACK

With our threat model defined, we can now discuss the mechanics of our attack on the model's parameters. Parameter attacks are similar to input attacks on images. The key difference is the parameters of the model and not the pixels of the image are being altered. Two broad categories of parameter attacks are gradient descent-based attacks and a single-bias attack (SBA). The gradient descent attack is similar to the FGSM attack [8]. Again, the key difference is the perturbations are added to the weights of the model instead of the pixels in the adversarial example. SBAs work very similarly to a "single-pixel" attack [46]; but instead of perturbing a single pixel in an input image, an SBA perturbs a single parameter of the model [9]. We chose to demonstrate supply chain risk using an SBA, because they are simpler to understand in a demonstration. Additionally, for simplicity of the demonstration, we limit out attack to the output layer.

An SBA takes advantage of the fact that we can force the output of a DNN classifier with a ReLu like activation function to be class i by adding a sufficiently large bias to the ith output neuron [9]. For the bias added to the ith neuron to be sufficiently large, the DNN classifier's output must converge upon the ith class [9].

There are two facets to this attack: effectiveness and stealth [9]. We define the effectiveness of an attack to be the probability of misclassifying the target class of images. We also adopt the definition of [9], for stealth of our attack. We represent the effectiveness and stealth of our attack through a custom loss function as seen in the following equation.

$$(1) \qquad Loss = M + L$$

where M is the probability of misclassifying the target class of images, and L is the absolute value of the change in accuracy of the classifier

Now that we have a means of measuring the performance of our attack, we can describe the attack. The first step in the attack is determining the sink class for the bias of each neuron. The sink class of the bias of a neuron is the output of the DNN classifier when a sufficiently large bias is added to the neuron [9]. To determine which neuron is biased towards the sink class, we follow the steps in Equation 2. First, initialize all neurons to 0. Second, add a bias value to a neuron. The output of DNN classifier will be the sink class of the neuron with the bias value. Follow this process for every neuron to determine the sink class of each neuron. Because our attack is limited to the output layer, this step is unnecessary. A large bias added to the 0th output neuron will force output to converge on the class of 0, the 1st output neuron with a large bias forces the output to converge on the class of 1, and so forth.

$$(2) \qquad \text{Sink class of } \theta_n = f(\theta_n + \varepsilon)$$

where $f(\theta)$ is the output of a DNN classifer with all output neurons initialized to 0, and ε is a bias value of 1

Once the sink classes have been established, the second step in the SBA is determining the size of a bias needed to cause the DNN classifier to converge on the targeted sink class [9]. This needed bias value can be obtained by the following the optimization problem defined by Equation 3.

$$\text{(3)} \quad \text{maximize } f_{adv}(\theta_n + \varepsilon)$$
$$w.r.t. |\varepsilon| \leq L$$

where $f_{adv}(\theta)$ *is the probability of classifying input as the target sink class, and* ε *is a bias value.*

This will result in a DNN that converges on a single target class. One of the previously stated goals of our attack is stealth. This attack is not stealthy because it will result in a classifier whose output converges upon a single target class, and it more closely resembles a Denial of Service (DOS) attack [9], as the performance of the classifier will collapse to the fraction of test data in the targeted class. We want to be able to misclassify a target class without misclassifying the untargeted classes. In our attack, we modify the original SBA by formulating the related optimization problem, as seen in Equation 4.

$$\text{(4)} \quad \text{minimize } f_{adv}(\theta_n - \varepsilon)$$
$$w.r.t. |\varepsilon| \leq L$$

where $f_{adv}(\theta)$ *is the probability of classifying input as the target sink class, and* ε *is a bias value.*

Our attack forces a DNN classifier with a ReLu like activation function to **not** be class i by **subtracting** a significantly large bias from the ith output neuron. For the bias subtracted from the ith output neuron to be sufficiently large, the DNN classifier's output must **diverge** from the ith class. This will result in the DNN classifier not classifying the targeted class as the correct class without affecting the classifier's classification of the untargeted classes.

On a practical side note, mounting a successful attack does not require a precise bias value [9]. We simply need a bias that is large in comparison to the other output neurons [9]. The upper bound of L in our equation is there to help avoid anomaly detection techniques. In our attack we arbitrarily chose a value of 1000 as the upper limit of our bias. This large number would stand out as an anomaly in a visual inspection of our model's weights, but it ensures in our demonstration that the bias is sufficiently large compared to the other output neurons.

C. MAINTAINING MODEL INTEGRITY

Vulnerability to the SBA in our setting results from trust in the model's dependencies, in this case the ability of the third-party platform to maintain data integrity against unauthorized parties. By assuming that the model on the third-party platform at any moment is the intended model, we become vulnerable if the model has been altered without our knowledge. This breach is not a result of a vulnerability of the third-party platform, but of the fact that integrity of the model is not assured across our supply chain. We need a defense that can detect when our model has been compromised as it advances through its life cycle. As discussed in the previous chapters, strategic use of security controls, as seen in Figure 11, are a way to detect and prevent the spread of compromises in the supply chain.

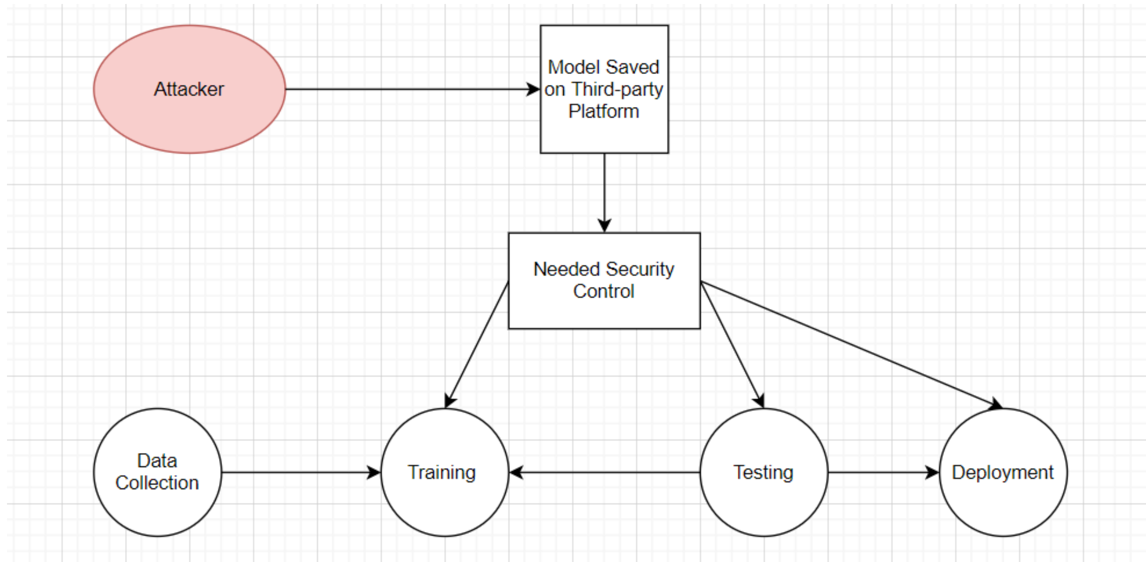

Figure 11. Needed Checkpoint. Source: [27]

We need a security control between life cycle phases to verify the model we are loading from the TensorFlow checkpoint is the model that we previously saved during training. There are several ways we can craft such a checkpoint. There have been multiple methods proposed on how to detect if a model's parameters have been altered [12]–[14]. These methods work, but they are computationally expensive and needlessly complicated. A simpler technique would be to save the integrity metadata of the model's parameters to

the assumed secure file system outside of the third-party platform upon completion of training and then upon future access of the model to compare the copies of the integrity metadata. Saving integrity data to an off-site file system does not completely mitigate the integrity concerns of our model. It simply moves integrity concerns to another location; however, it is desirable because it is an added layer of defense and verifying the integrity of metadata is computationally cheaper than verifying the integrity of the entire model through one of the previously researched techniques [12]–[14]. The proposed defense technique is represented in Figure 12.

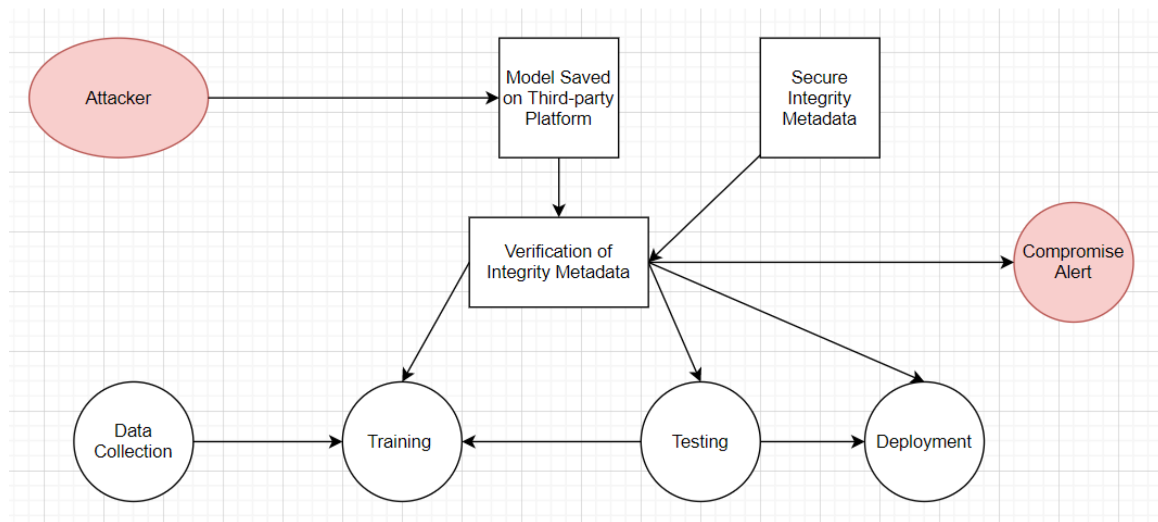

Figure 12. Our Checkpoint. Source: [27]

Upon accessing the saved model, we would compare the off-site saved integrity metadata with the integrity metadata of the model that were downloaded from the TensorFlow's checkpoint saved on third-party platform. If the integrity metadata match, we can have more confidence that an attacker has not altered our model stored on the third-party platform. If the integrity metadata do not match, we are alerted to the breach and can address the alert before proceeding to the next phase.

This is simple and elegant. It can detect SBAs and other attacks on model parameters. Because our checkpoint utilizes cryptographically secure hashes and hashes give fixed-size outputs and cryptographically secure hash functions have preimage and

second preimage resistance, verification of the integrity metadata will fail in the same way if one parameter is changed or if every parameter of the model is changed. This mitigates our reliance on a third-party to maintain data integrity over time. Although this solution is straightforward, we have not seen it described in literature. Implementation details of the machine learning application, attack mechanics, and defense mechanics are discussed in the next subsection.

D. IMPLEMENTATION

The machine learning application for this experiment will be a machine learning model that classifies images. The workflow for this application will follow the static machine learning workflow, as presented in Figure 6. There are four phases: Data Collection, Training, Testing, and Deployment. Additionally, there are dependencies for each phase.

In the data collection phase, we use the Modified National Institute of Standards and Technology (MNIST) database. The MNIST database is very popular for creating simple machine learning classification models. The database contains 60,000 training images and 10,000 testing images. Each image is 28 by 28 pixels and conforms to grayscale levels. We download the database from the built-in MNIST dataset in the TensorFlow Application Programming Interface (API). Examples from the dataset are provided in Figure 13.

Figure 13. MNIST Dataset Examples. Source: [44].

Because of the popularity of third-party software in machine learning development, we utilize Google's Colab in our data collection, training, testing, and deployment phases. Colab is a free Jupyter notebook environment. The notebooks are stored on Google Drive, and they are run in the cloud. Colab supports many programming languages; but in this experiment, we exclusively use Python3. Additionally, we use the following libraries in the creation and testing of our model TensorFlow, Keras, NumPy, and OS. These libraries enjoy widespread popularity among machine learning developers.

We begin by importing the necessary libraries: TensorFlow, Keras, NumPy, and OS. Next, we download the MNIST dataset. Additionally, we clean and format the data through TensorFlow's `reshape` method. We select an Adam optimizer and a ReLu activation function. The model is made up of four layers: input, dense layer, dropout layer, and another dense layer. We want to be able to access our model upon later dates, so we save all parameters of our model to the TensorFlow checkpoint at the end of the training phase. A visual representation of our workflow up to this point is depicted in Figure 14.

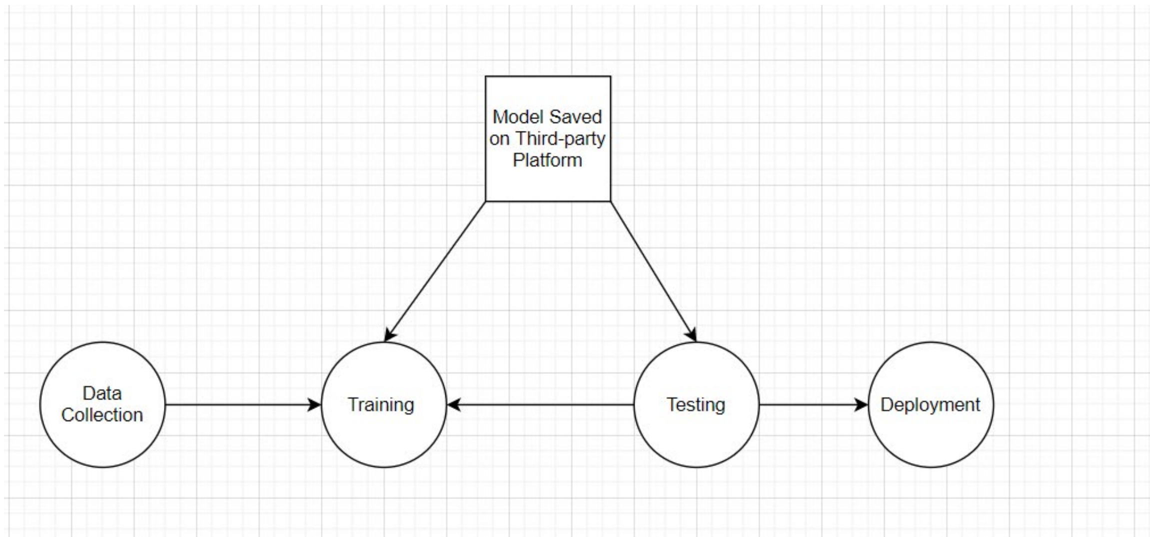

Figure 14. Model Supply Chain. Source: [27].

There are often multiple training and testing iterations in the creation of the model, so there is an added path from testing to the training phase.

When constructing our secure checkpoint, we wrap the standard TensorFlow checkpoint API calls to make a new custom checkpoint. Within out custom checkpoint, we compute the hash of the saved model's parameters with a SHA-256 algorithm and save it to the assumed secure off-site file system.

When constructing the attack, we used two custom functions to implement Equations 2 and 3. The function determined the sink classes of the output neurons, and the second optimized our bias value. After creating the baseline model, defense technique, and attack, our workflow is represented by the workflow depicted in Figure 14.

E. EXPERIMENTAL RESULTS

Before we can examine the effectiveness of our attack or defense, we need to establish the baseline performance of our model. We will use overall accuracy as the performance metric for the base model. The accuracy was calculated by using TensorFlow's `predict` method. The accuracy of the model with 10 epochs was 98.2%.

With the performance of the base model established, we will first look at the performance of the attack and then at the performance of the defense. To test the

performance of our single bias attack, we ran the attack a total of 10 times, once for each class of images. We describe the performance of the attack in terms of a custom loss function, as described in Equation 1. This function has two terms: the change in accuracy of the model and the probability of misclassifying the selected class of images. We will examine each of these terms individually in Figures 15–17 before examining the overall loss itself in Figure 18.

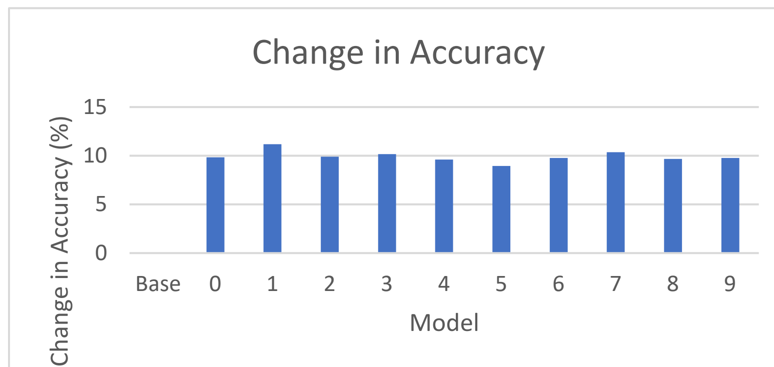

Figure 15. Change in Accuracy versus Target Class

The change in accuracy of the altered models ranged from 9.0% to 11.18%. At first glance, this appears to be an unacceptably large degradation in the performance of the model; however, this large change in accuracy is a result of the relatively few output neurons. Looking at Figure 16, we can see that each class of images makes up approximately 10% of the data. If we compare the change in accuracy of the altered model with the percentage of the misclassified class of images, we see that they roughly correspond. The small difference between change in accuracy and percentage of the misclassified images is a result of the imperfect accuracy of our baseline model.

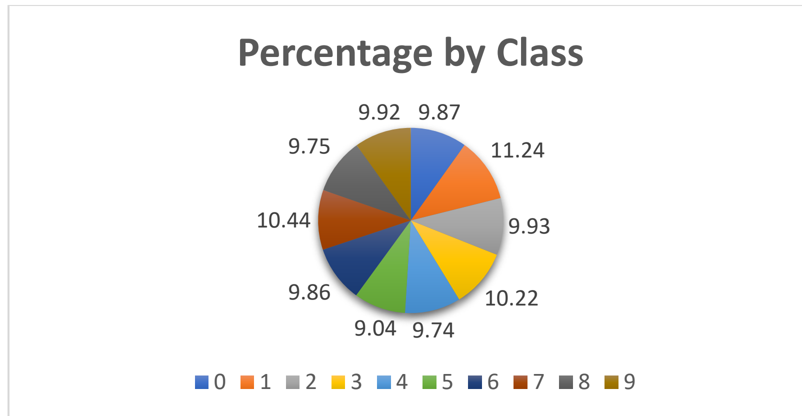

Figure 16. Dataset Composition by Image Class (percentage)

Next, we will look at Figure 17, which represents the probability of misclassifying the selected image. The baseline model had roughly a 0.2% chance of misclassifying a selected image, because of the inherent 98.2% accuracy of the model. For all 10 classes of images, the altered model had 100% probability of misclassifying the selected class of images.

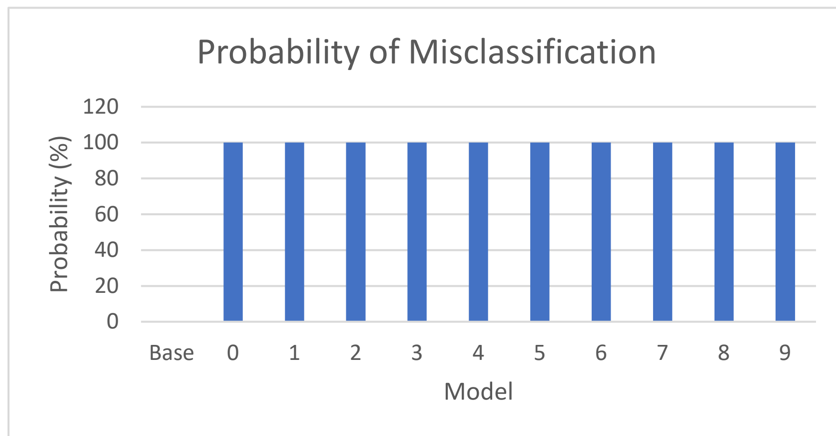

Figure 17. Probability of Misclassification versus. Target Class

Now that we have looked at the individual parts of the loss function. We will examine the loss of each altered model. The best possible loss value would be 100% probability of misclassification and 0% degradation of model accuracy. The loss of the altered models represented in Figure 18 were in the range of 0.89 and 0.91, which is very

close to the maximum possible loss value; and again, to emphasize the point of Figure 16, an expected loss of a well-functioning attack would approximately be 0.9, given that each targeted misclassified class of images was approximately 10% of the total images.

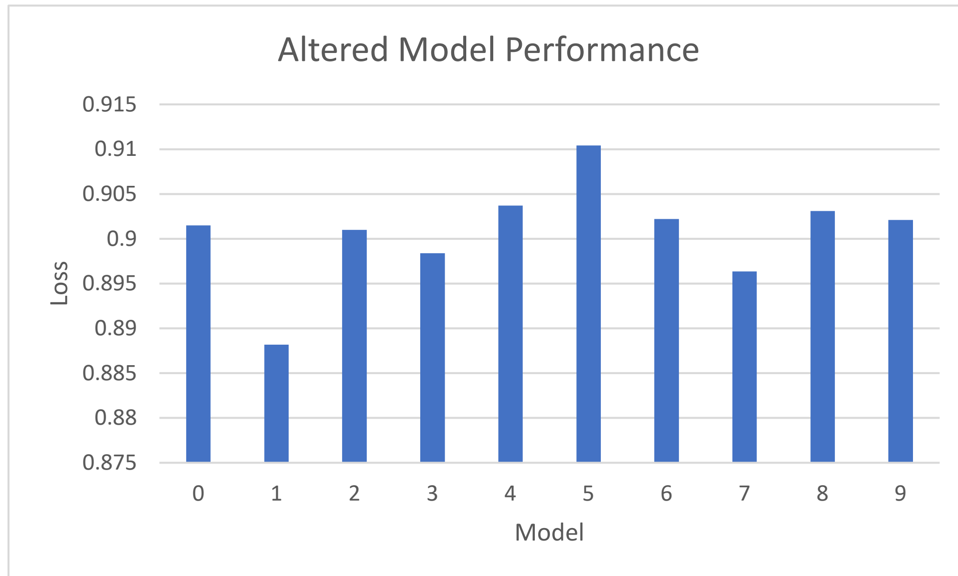

Figure 18. Attack Performance Using Custom Loss versus Target Class

These results show that our single bias attack is effective enough to be an efficient attack and sufficient for our demonstration purposes, but it would not be stealthy enough to attack a classifier with a relatively few output neurons. Next, we will examine the effectiveness of our proposed defense.

After implementing our defense, we used our base-line classifier to predict the class of images in the test images dataset and then recorded the output of our custom checkpoint. Next, we used our SBA to attack an output neuron, used the altered classifier to predict the classes of the images in the test dataset, and then recorded the output of our custom checkpoint. We did this for a total of 10 attacks, once for every output neuron of our classifier. Our results are seen in Figure 19. Of the 11 times our custom checkpoint was used, we recorded a total of 10 true-positives and 1 true-negative values.

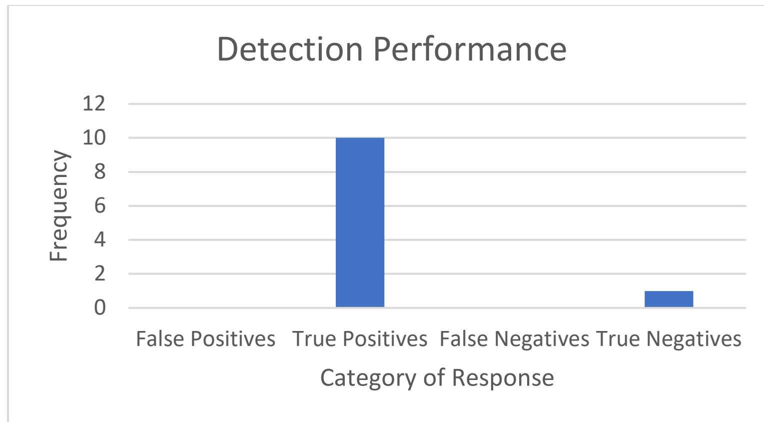

Figure 19. Detection Performance

We will discuss the implications of our results in Chapter IV.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSION

As stated throughout this thesis, the purpose of this research is not to demonstrate a new type of supply chain attack or a new defense technique. The purpose is to demonstrate that the machine learning supply chain is vulnerable to a specific supply chain attack and how security controls like our proposed secure ML checkpoint are a viable technique to mitigate a broad class of supply chain attack. Machine learning supply chains are vulnerable to supply chain attacks because of the lack of sufficient industry security standards and the large amount of trust put in dependencies [4]. Specifically, the outsourcing of the training and testing of the model to third-party platforms such as AWS or Colab increases the trusted computing base in an ML model. Because software supply chains tend to be highly co-dependent and managed in active, cyclical ways, compromises in any phase of the of the supply chain or in the one of the dependencies can affect the rest of the supply chain.

Additionally, it is impossible to have complete trust in the security of each phase and dependency of the supply chain [38], [39]. Because of this inherent risk, when implementing security within a supply chain, we need a design with possible failure in mind and the ability to detect an attack. We believe that strategically placed security controls such as our proposed secure checkpoints can identify compromises within a phase or dependency before proceeding to the rest of the supply chain.

A. CONTRIBUTION

To examine the validity of this claim, we created our own machine learning application with its own supply chain. We showed how an attacker with access to the third-party platform on which a model is developed can exploit our trust in the third-party platform by attacking our saved model through an SBA. We demonstrated the successful misclassification of an entire class of images by subtracting a large arbitrary bias from a targeted output neuron. We then demonstrated how to construct a secure ML checkpoint by extending the standard TensorFlow checkpoint API. Our secure checkpoint was effective at alerting us to the presence of an SBA. This check point used cryptographic

hashes to implement a requirement for checkable integrity metadata. While not as sophisticated or complicated as some other detection methods [12]–[14], our checkpoint was not computationally expensive and performed with only true-positives and true-negatives. To our knowledge, there is no mention in literature of using integrity verification to protect machine learning model integrity.

While our machine learning application was trivial, it is sufficient for example purposes. Many machine learning applications are more complex but could be defended in precisely the same manner. The compromise of critical application such as facial recognition or image classification of weapons systems could have devastating consequences. Organizations need to know that the integrity of their machine learning model has not been compromised. We were able to demonstrate how easily a machine learning model can be compromised, and how such an attack can be detected. It is crucial to the success of an organization to implement security controls at strategic points in their supply chain. We hope that our research will be the catalyst for organizations to be proactive in their defense of the supply chain.

B. FUTURE RESEARCH

Our proposal for a secure checkpoint provides a reliable defense against a class of supply chain attacks, but it is a fairly blunt approach that could be improved with future research. For example, our current checkpoint simply alerts us to the presence of modifications our model. Our checkpoint does not allow us to see which of the output neurons were perturbed. Training machine learning models is expensive. It could be valuable if the perturbations were limited to a specific neuron, and developers were able to salvage the model. Finer grained constructions of the required integrity metadata, such as the use of Merkle trees would enable more detailed descriptions of the detected model perturbations to be produced efficiently (models can involve millions or billions of parameters, making direct comparisons inefficient or rendering saving the full model to secure storage impractical).

Additionally, our proposed checkpoint usefulness is limited in scope. It works only on static machine learning models. Online machine learning models that are constantly

updating parameters with incoming feedback would not be compatible with our approach. A different hash would constantly be calculated with every update of the model rendering our checkpoint useless. Future research could develop integrity management methods compatible with online machine learning models.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] M. Willett, "Lessons of the SolarWinds hack," *Glob. Polit. Strategy*, vol. 63, no. 2, pp. 7–26, Mar. 2021.

[2] A. Greenberg, "Hacker lexicon: What is a supply chain attack?," *Wired*, May 2021, [Online]. Available: https://www.wired.com/story/hacker-lexicon-what-is-a-supply-chain-attack/

[3] K. Thompson, "Reflections on trusting trust," presented at the Turing Award Ceremony, San Francisco, CA, 1984.

[4] R. Kumar et al., "Adversarial machine learning - industry perspectives," *SSRN Electron. J.*, 2020, [Online]. Available: https://www.semanticscholar.org/paper/Adversarial-Machine-Learning-Industry-Perspectives-Kumar-Nystr%C3%B6m/3eb594bdc7057858a7bcd6243947c1944e89e2e3

[5] L. Liang et al., "Deep Sniffer: a DNN model extraction framework based on learning architectural hints," presented at the ASPLO's 20, Lausanne, Switzerland, 2020.

[6] M. Yan, W. Fletcher, and J. Torrellas, "Cache telepathy: leveraging shared resource attacks to learn DNN architectures," presented at the 29th USENIX Security Symposium, San Diego, CA, Aug. 2020.

[7] M. Goldblum et al., "Dataset security for machine learning: data poisoning, backdoor attacks, and defenses," *arXiv*, Dec. 2020, [Online]. Available: https://arxiv.org/abs/2012.10544

[8] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," presented at the ICLR, Mountain View, 2015.

[9] Y. Liu, L. Weu, B. Luo, and Q. Xu, "Fault injection attack on deep neural network - IEEE Conference Publication," presented at the IEEE International Conference on Computer-Aided Design, Irvine,CA, Nov. 2017.

[10] J. Clements and Y. Lao, "Hardware trojan attacks on neural networks," *arXiv*, Jun. 2018, [Online]. Available: https://arxiv.org/abs/1806.05768

[11] M. Zou, Y. Shi, C. Wang, F. Li, W. Song, and Y. Wang, "PoTrojan: powerful neural-level trojan designs in deep learning models," *arXiv*, Feb. 2018, [Online]. Available: https://arxiv.org/abs/1802.03043

[12] Y. Liu, W. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, “ABS: scanning neural networks for back-doors by artificial brain stimulation,” in *ABS: Scanning Neural Networks for Back-doors by Artificial Brain Stimulation*, 2019, pp. 1265–1282.

[13] X. Xu, Q. Wang, H. Li, N. Borisov, N. Gunter, and B. Li, “Detecting AI trojans using meta neural analysis,” *arXiv*, Oct. 2019, [Online]. Available: https://arxiv.org/abs/1910.03137

[14] K. Liu, B. Dolan-Gavitt, and S. Garg, “Fine-Pruning: defending against backdooring attacks on deep neural networks,” *arXiv*, May 2018, [Online]. Available: https://arxiv.org/abs/1805.12185

[15] C. Perrow, “Normal accidents,” *Organ. Environ.*, Mar. 2004, Accessed: Jul. 28, 2021. [Online]. Available: https://journals.sagepub.com/doi/pdf/10.1177/1086026603262028?casa_token=q4j3jshqI_sAAAAA%3AThQqiN_8SfuzoI9Hw6mUxeJVaPw39bL7RIQhvjwa0VTetD6HzBuB2tNVbPMNEv-_oKX2P8XEC7HV&

[16] H. Bonner, “Efficient vs. responsive supply chain or can you have both?,” *Efficient vs. Responsive Supply Chain or Can You Have Both*, Sep. 22, 2020. https://riskpulse.com/blog/efficient-vs-responsive-supply-chain/ (accessed Aug. 06, 2021).

[17] T. Randall, M. Ruskin, and A. Morton, “Efficient versus responsive supply chain choice: an empirical examination of influential factors,” Taylor R. Randall, Ruskin M. Morgan, Alysse R. Morton,” J. *Prod. Innov. Manag.*, vol. 20, no. 6, pp. 430–443, Nov. 2003.

[18] R. Hawkins and P. Ballon, “When standards become business models: reinterpreting ‘failure’ in the standardization paradigm,” *Info*, vol. 9, no. 5, pp. 20–30.

[19] Magaya, “Six supply chain models you need to know,” *Industry*,. https://info.magaya.com/blog/the-6-degrees-of-supply-chain-connections (accessed Aug. 06, 2021).

[20] Inflectra, “What is agile software development?,” *Inflectra*. https://www.inflectra.com/methodologies/agile-development.aspx (accessed Aug. 06, 2021).

[21] K. Beck et al., *Manifesto for Agile software development*. Accessed: Jul. 16, 2021. [Online]. Available: https://moodle2019-20.ua.es/moodle/pluginfile.php/2213/mod_resource/content/2/agile-manifesto.pdf

[22] J. Hoek, “Pursuing a full agile software development life,” Mendix, May 16, 2018. https://www.mendix.com/blog/pursuing-a-full-agile-software-life cycle/#blogAuthor (accessed Aug. 25, 2021). [1]

[23] O. Anuirina, "Agile SDLC: skyrocketing your project with agile principles," Mlsdev, Aug. 25, 2021. https://mlsdev.com/blog/agile-sdlc (accessed Aug. 25, 2021).

[24] M. Shi, "Software functional testing from the perspective of business practice," *Comput. Inf. Sci.*, vol. 3, no. 4, pp. 49–52, 2010.

[25] "Acceptance testing for software as a service (SaaS)," *Softw. Qual. Prof.*, vol. 23, no. 1, pp. 4–10, Dec. 2020.

[26] *Inflectra*, "What is scaled Agile?" https://www.inflectra.com/methodologies/scaled-agile.aspx (accessed Aug. 25, 2021).

[27] A. Pelz-Sharpe and K. Kompella, "This is why you're approaching your AI project wrong," *Document Strategy*, Mar. 2019. https://documentmedia.com/article-2896-This-Is-Why-Youre-Approaching-Your-AI-Project-Wrong.html (accessed Aug. 25, 2021).

[28] M. Finnemore and D. Hollis, "Constructing norms for global cybersecurity," *Am. J. Int. Law*, vol. 110, no. 3, pp. 425–279, Jul. 2016.

[29] D. Longley and M. Shain, *Data & Computer Security, Dictionary of Standard Concepts and Terms*. London: Palgrave Macmillan. Accessed: Aug. 02, 2021. [Online]. Available: https://link-springer-com.libproxy.nps.edu/content/pdf/10.1007%2F978-1-349-11170-1.pdf

[30] R. Hofferbert, "Social science archives and confidentiality," *Am. Behav. Sci.*, vol. 19, no. 4, p. 467, 1976.

[31] D. Clark and D. Wilson, "A comparison of commercial and military computer security procedures," *in A Comparison of Commercial and Military Computer Security Policies*, Oakland, CA, pp. 184–193. Accessed: Aug. 02, 2021. [Online]. Available: https://www-proquest-com.libproxy.nps.edu/docview/194627188?pq-origsite=primo

[32] K. J. Biba, "Integrity considerations for secure computer systems," *Air Force*, Bedord, Massachusetts, ADA039324, 1977. Accessed: Aug. 27, 2021. [Online]. Available: https://apps.dtic.mil/sti/pdfs/ADA039324.pdf

[33] Comtact Ltd., "What is the CIA triad?," Feb. 07, 2019. https://comtact.co.uk/blog/what-is-the-cia-triad/ (accessed Aug. 25, 2021).

[34] M. Faisal, A. Ikram, K. Muhammad, S. Min, and J. Kim, "Establishment of trust in internet of things by integrating trusted platform m: *To Counter Cybersecurity Challenges*," Complexity, vol. 2020, 2020, Accessed: Aug. 05, 2021. [Online]. Available: https://www-proquest-com.libproxy.nps.edu/docview/2484137471?pq-origsite=primo

[35] L. Futcher and r von Solms, "SecSDM: a model for integrating security into the software development life cycle," presented at the Fifth World Conference on Information Security Education, West Point, NY, 2007.

[36] S. Eggers, "A novel approach for analyzing the nuclear supply chain cyber-attack surface," *Nucl. Eng. Technol.*, vol. 53, no. 3, pp. 879–887, Sep. 2020.

[37] National Computer Security Center, *Computer Security Subsystem Interpretation of the Trusted Computer System Evaluation Criteria*. Fort George G. Meade, MD : The Center ; Washington, D.C.? : Supt. of Docs., U.S. G.P.O., distributor, 1988.

[38] M. Harrison, W. Ruzzo, and J. Ullman, "On protection in operating systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 9, no. 5, pp. 14–24, Nov. 1976.

[39] Alan Turing, "On computable numbers, with an application to the Entscheidungs Problem," in Proceedings of the London Mathematical Society, vol. s2-42, pp. 230–265. Accessed: Aug. 03, 2021. [Online]. Available: https://academic.oup.com/plms/article/s2-42/1/230/1491926

[40] J. B. Michael and J. Viega, "Struggling with supply-chain security," *Computer*, vol. 54, no. 7, pp. 98–104, Jul. 2021.

[41] M. Faisal, A. Ikram, K. Muhammad, S. Min, and J. Kim, "Establishment of trust in internet of things by integrating trusted platform module: to counter cybersecurity challenges," *Complexity*, vol. 2020, 2020, Accessed: Aug. 05, 2021. [Online]. Available: https://www-proquest-com.libproxy.nps.edu/docview/2484137471?pq-origsite=primo

[42] M. Puliparambil, M. Sindhu, S. Chungath, and S. Madathil, "Hash-one: a lightweight cryptographic hash function," *IET Inf. Secur.*, vol. 10, no. 5, pp. 225–231.

[43] S. Raza, S. Duquennoy, J. Hoglund, U. Roedig, and T. Voigt, "Secure communication for the internet of things--a comparison of link-layer security and IPsec for 6LoWPAN," *Secur. Commun. Netw.*, vol. 7, no. 12, pp. 2654–2668.

[44] J. Steppan, *MnistExamples*. 2017. Accessed: Aug. 16, 2021. [Online]. Available: https://commons.wikimedia.org/wiki/File:MnistExamples.png

[45] D. Villalva, J. Onaolapo, G. Stringhini, and M. Musolesi, "Under and over the surface: a comparison of the use of leaked account credentials in the dark and surface web," *Crime Science*, vol. 7, no. 1, pp. 1–11, Nov. 2018.

[46] J. Su, D. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, Oct. 2019.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California