International Journal of Engineering in Computer Science

E-ISSN: 2663-3590
P-ISSN: 2663-3582
www.computersciencejournals.com/ijecs
IJECS 2024; 6(2): 101-105
Received: 14-05-2024
Accepted: 28-06-2024

SV Ramana
Assistant Professor, Department of CSE, Malla Reddy Engineering College for Women, Autonomous, Hyderabad, Telangana, India

E Bindu
Student, Department of CSE, Malla Reddy Engineering College for Women, Autonomous, Hyderabad, Telangana, India

K Prasanna Rao
Student, Department of CSE, Malla Reddy Engineering College for Women, Autonomous, Hyderabad, Telangana, India

K Chandana
Student, Department of CSE, Malla Reddy Engineering College for Women, Autonomous, Hyderabad, Telangana, India

Corresponding Author:
SV Ramana
Assistant Professor, Department of CSE, Malla Reddy Engineering College for Women, Autonomous, Hyderabad, Telangana, India

Using ensemble machine learning algorithm for applications in recognizing false data injection attacks and an effective privacy-improving in smart grid

SV Ramana, E Bindu, K Prasanna Rao and K Chandana

DOI: https://doi.org/10.33545/26633582.2024.v6.i2b.130

Abstract
One well-known machine learning paradigm, federated learning (FL) assists with data privacy by letting clients save raw data locally and sending only native parameters for the model to a data aggregator server to build a shared global model. Unfortunately, federated learning may be hacked by unscrupulous aggregators who use model parameters to deduce customers' training data. Most of the existing solutions to this problem rely on a non-colluded server setup, rely on a trusted third party to calculate master secret keys, or use a safe multiparty computation protocol, none of which improve efficiency when applied to repeated computations of an aggregate model. We provide a privacy-preserving cross-silo federated learning technique that is both efficient and secure. An effective privacy-preserving federated educational protocol is achieved by utilizing secret sharing only during the establishment phase and iterations if parties re-join, and by accelerating.
Computation achievement via parallel computing. Our double-layer encryption Scheme does not require computing discrete logarithm. Additionally, clients are allowed to drop out and re-join throughout the training process. Theoretically and experimentally, the suggested technique achieves acceptable model utilities while providing proved anonymity vs. an honest-but-curious aggregator server. The method is implemented in smart grids for the purpose of detecting fake data injection attacks (FDIA). This outshines previous efforts by providing a safe method of cross-silo FDIA federated training that is resistant to assaults on local private data inference.

Keywords: Smart grid, false data injection attacks (FDIA), federated learning (FL)

Introduction

A new machine learning paradigm called federated learning allows clients to keep the initial data locally and only send the modified local model settings to an aggregator server so they may train a global model together. This solves important data privacy problems. This feature is what makes federated learning a better privacy option than consolidating all training data into one place. One potential issue with federated learning is the possibility of inference attacks. These attacks allow dishonest aggregators to potentially learn clients' training data by analysing their model parameters, such as weights and gradients. Used in generative adversarial networks for deducing a target client's private data from shared model parameters is one example. So, data privacy cannot be assured with any degree of rigor even when a model is trained via federated learning. Since data sample are anonymized across numerous clients, it is possible to extract information from world model parameters without associating it with a particular client. But this won't work if the data is derived from faulty aggregates that are based on local model parameters. To avoid these inference attacks, it is important to prevent a corrupt aggregator from accessing the model parameters of clients. Current methods for dealing with this issue primarily use two approaches: secure aggregation and differential privacy. The former has the disadvantage of compromising global model accuracy for privacy-utility by directly adding noise to the client's models across a large number of iterations. The latter secretly aggregates the customers' models without revealing their exact values by using cryptographic methods like homomorphic encryption and safe multiparty computing. Current methods for aggregation master key generation often include a trusted outsider or use a scenario with several non-colluding servers. In addition, the high cost of computation and interaction between several clients across several training cycles makes many of the suggested systems unworkable and inefficient. As a vital security operation, detecting false data injection attacks (FDIAs) is essential for smart grid control systems. This was resolved using techniques of data-driven machine learning. Distributed throughout an interconnected grid, the massive amounts of measurement data are necessary for the information-driven machine learning algorithms. Regarding the deregulation of the electricity sector, with this kind of interconnected arrangement, each sub-grid is owned and operated by a separate transmission grid corporation (TGC). Sharing the measurements from all sub-grids involved is essential for building an accurate model for fake data injection detection. This massive amount of measurement data, however, cannot be sent via a Volume: 18, Issue Date: 17. April 2023 of the IEEE Transactions on Data Forensics and Security using Machine Learning In addition to being costly, a centrally managed detection machine learning system might cause privacy and security problems, such as competitive privacy. The issue is how to keep their competitive privacy while coordinating these TGCs to identify FDI assaults. Recent research have focused on federated learning-based solutions to this complex challenge. A typical scenario in federated learning is a cross-silo environment where many organizations or businesses share the goal of training a model using all of their data but are hesitant to share it directly with each other owing to privacy, security, or legal concerns. An effective privacy-preserving cross-silo federated learning system for FDIA detection across multi-area transmission grids is necessary to protect the privacy of power firms when they provide their local training models. Considering these points, we provide a solution that may be used in the smart grid domain: quick cross-silo supervised learning with robust privacy protection. We establish a successful privacy-preserving federated education protocol by constructing a double-layer encrypting scheme spanning several federation learning rounds and leveraging Shamir secret sharing. This protocol also enables certain clients to drop out and then re-join dynamically throughout the training process. To be more precise, here are the key points: The creation of a secure weighted aggregation technique for universal cross-silo federated learning that enhances privacy is based on Shamir secret sharing and lightweight double-layer encryption. The approach gets over the problem that several comparable studies have, which is that you have to compute discrete logarithms. It is not necessary to have many non-colluding server configurations. Furthermore, decentralization helps in increasing privacy by generating the secret keys of the two encryption levels used by clients. Theoretically and experimentally, the suggested technique achieves acceptable model usefulness while providing proved privacy versus an honest-but-curious aggregator server. The suggested method is resistant to training iterations in which participants leave out or re-join, and it is fast in communication and computing. In order to effectively identify FDIAs in the smart grid domain, we present and experimentally test a cross-silo federated learning system that improves privacy and is resistant to assaults on inference from local training data. The document is structured into eight parts. The sections on Preliminaries and Related Works follow this Introduction. Section 4 presents the suggested privacy-enhancing cross-silo federated education system, which does not involve any trusted third parties. Section 5 then analyses the scheme. Sections 6 and 7 provide an empirically evaluated realistic scenario of improving privacy in cross-silo training for FDIA identification in smart grids. Lastly, the arguments and conclusions are presented in Section 8.

Related Work

Investigating Interconnections in the Enron Email Database

Researchers are interested in the Enron emails corpus for three reasons: (a) it is a massive collection of emails from a genuine company; (b) it spans three and a half years; and (c) it is accessible. We add to the first social network analytics-based examination of the Enron emails dataset in this article. In this paper, we detail our efforts to improve the Enron corpus by adding relational data and removing communication networks. In order to discover important participants across time and investigate the structural features of Enron's networks, we use a number of network analytic methods. The network was denser, more centralized, and more linked under the Enron crisis than it is during normal times, according to our early data. According to our data, there was greater cross-functional contact among Enron workers throughout the crisis, regardless of employees' official roles. However, the top executives remained close-knit, supported each other, and interacted with the remainder of the business via extensive brokering. Organizational crisis scenario modelling and failure indicator research stand to profit from the insights obtained via the studies we conduct and suggest.

Proof that metrics for object-oriented design are reliable measures of quality

This article details the findings of an investigation into the oriented toward objects (OO) design metrics proposed in (Chid amber with Kemmerer, 1994) and their practical application. In particular, we want to find out whether these measures may be employed as initial quality indicators by evaluating them as predictors of classes that are prone to errors. This research supplements that of Li and Henry (1993), who also utilized the same set of criteria to evaluate the frequency of class maintenance modifications. For the purpose of our validation, we gathered information on the creation of eight information management systems for medium-sized businesses that met the same criteria. C++ and the famous OO analysis/design methodology were the tools of choice for all eight projects' development. The benefits and downsides of various OO measures are examined based on quantitative and empirical investigation. At the beginning of a class's lifecycle, a number of Chid amber with Kemmerer's OO metrics seem to be helpful in predicting the class's fault-proneness. Furthermore, when applied to our dataset, they outperform "traditional" code measures, which are only available later in the software development lifecycle, as predictors.

Rich: Rendering Integer-Based Vulnerabilities Safe by Design

An efficient tool for identifying arithmetic-based threats against programs written in C at run time, RICH (Run-time Arithmetic Checking) was designed and implemented by us. When a variable's value exceeds the range of the computer's word used to materialize it, a common programming mistake known as a C integer bug occurs [1-15]. For instance, while converting a big 32-bit integer to a smaller 16-bit short. We demonstrate that the well-known sub-typing theory captures both safe and dangerous integer operations in C. Compiling C programs into object code with the RICH compiler feature allows them to identify integer-based attacks by monitoring their own execution. After integrating RICH into the GCC compiler, we ran tests on several servers in the network and UNIX utilities. The operational penalty of RICH is quite modest, averaging approximately 5%, even though integer operations are ubiquitous. Two further integer problems were discovered by RICH, and of the ones we tested, all but one were detected. Based on these findings, RICH is an effective and lightweight tool for testing software and a defence mechanism for runtime. When programmers intentionally employ integer overflows, RICH could produce false positives. Additionally, as it does not represent all C features, it might overlook certain integer issues.

CSSV: Working Towards a Practical Method for Statically Identifying All C File Buffer Overflows

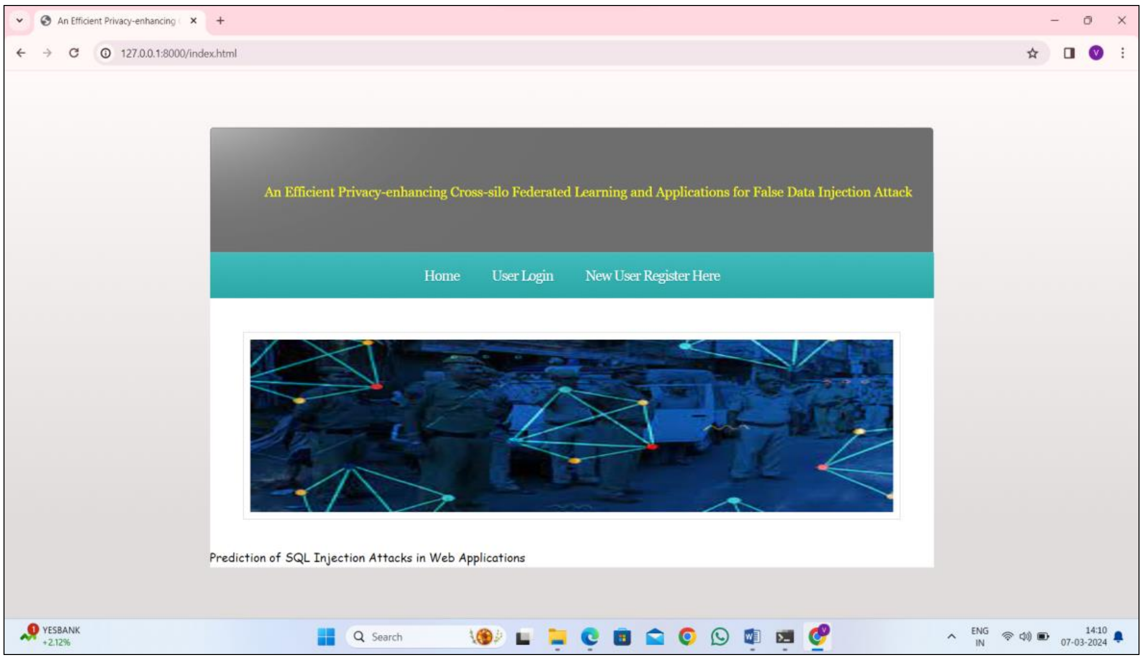
Software viruses sometimes take advantage of security holes in C programs caused by incorrect string manipulations. This program, called C Strings Static Verifier (CSSV), can statically detect any mistake made when manipulating strings. It detects all such mistakes, although it might occasionally generate false alerts since it is a cautious instrument. Thankfully, the stated false alert rate is low, demonstrating that it is possible to significantly decrease program vulnerability. CSSV analyses each operation independently, allowing it to manage massive applications. This is why the tool can validate procedure contracts. We built a CSSV prototypes and tested it against EADS Airbus's production code to ensure it was error-free. Applying CSSV to another popular string-intensive app revealed actual issues with few false positives. Expandable, lightweight static analysis for enhanced security

The vast majority of security breaches take advantage of common types of implementation defects. These issues occur with disturbing regularity despite developers' best efforts to identify and fix them before software deployment. This is not due to a lack of understanding of these vulnerabilities within the security community, but rather to the fact that methods for avoiding themselves have not been incorporated into software development. An extendable method for detecting common security vulnerabilities (such as format string vulnerabilities and buffer overflows) using lightweight static analysis is described in this article.

Methodology

1) **New User Registration:** Users who are new to the application may register.
2) **User Login:** One may access the application after signing up.
3) **Upload Dataset:** Once logged in, users can upload datasets to the program. From there, they can extract labels and queries. As an added step, they may stop words like "and," "or," and "what" from all searches. Core query words may be obtained by eliminating stop words from the application. By using the Natural Language Processing Toolkit, the core word dataset will be processed.
4) **Execute Ensemble Algorithms:** Prepare the dataset for training by feeding it into an Ensemble Machine Learning algorithm. Then, apply the trained model to test data and measure its accuracy and other metrics.
5) **Graph of Confusion Matrix:** This module will be used to draw a graph of the algorithm's capacity to forecast.
6) **Anticipate susceptibility:** An algorithm based on machine learning will examine all of the TEST data and make a prediction about the kind of vulnerability when this module is used to submit additional TEST data queries.

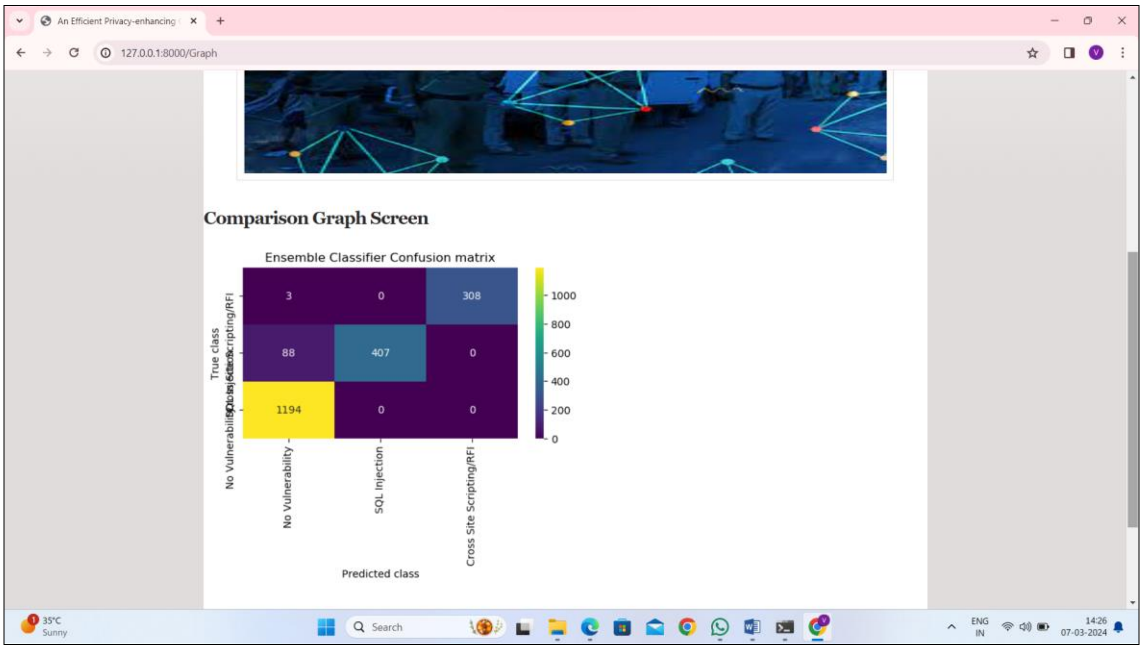
Results and Discussion



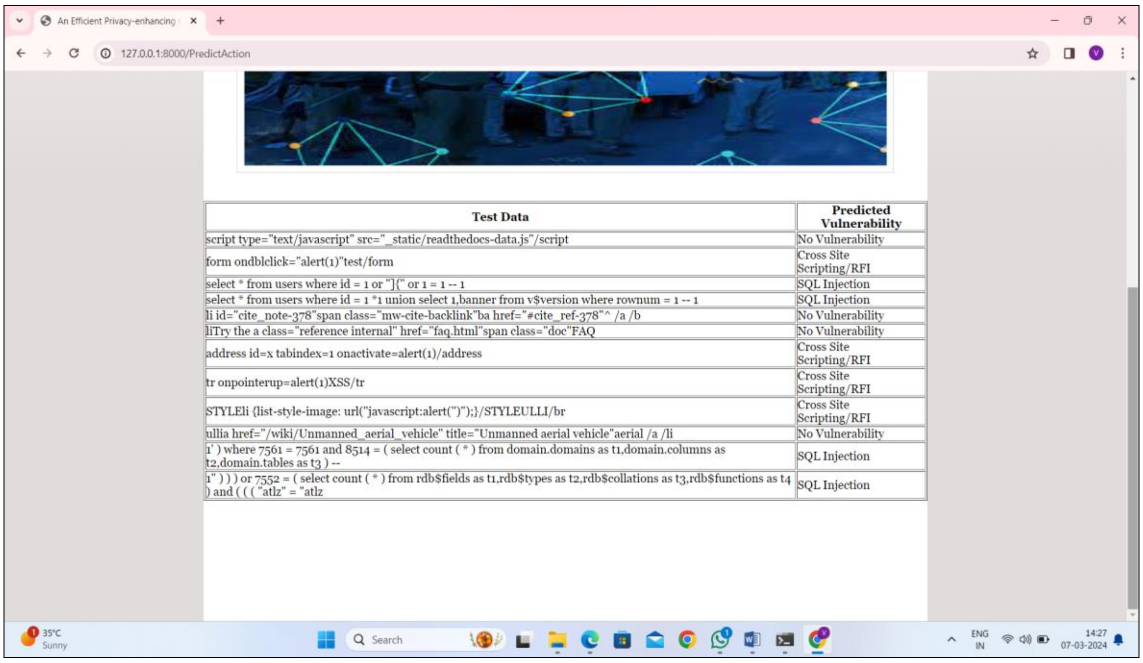
An Efficient Privacy-enhancing Cross-silo Federated Learning and Applications for False Data Injection Attack

Prediction of SQL Injection Attacks in Web Applications

To access the sign-up page, click on the "New User Register Here" link in the top result.



The x-axis in the above graph shows the predicted labels, the y-axis the true labels, and the various colored boxes on the diagonal show the number of accurate predictions, while the remaining blue boxes show the number of extremely few wrong predictions. After that, you may submit your test data and use the "Predict Vulnerability" option to make your vulnerability prediction.



Test Data	Predicted Vulnerability
script type="text/javascript" src="_static/readthedocs-data.js"/script	No Vulnerability
form ondblclick="alert(1)"test/form	Cross Site Scripting/RFI
select * from users where id = 1 or "]{" or 1 = 1 -- 1	SQL Injection
select * from users where id = 1 *1 union select 1,banner from v$version where rownum = 1 -- 1	SQL Injection
li id="cite_note-378"span class="mw-cite-backlink"ba href="#cite_ref-378"^ /a /b	No Vulnerability
liTry the a class="reference internal" href="faq.html"span class="doc"FAQ	No Vulnerability
address id=x tabindex=1 onactivate=alert(1)/address	Cross Site Scripting/RFI
tr onpointerup=alert(1)XSS/tr	Cross Site Scripting/RFI
STYLEli {list-style-image: url("javascript:alert(")");}/STYLEULLI/br	Cross Site Scripting/RFI
ullia href="/wiki/Unmanned_aerial_vehicle" title="Unmanned aerial vehicle"aerial /a /li	No Vulnerability
1') where 7561 = 7561 and 8514 = (select count (*) from domain.domains as t1,domain.columns as t2,domain.tables as t3) --	SQL Injection
1"))) or 7552 = (select count (*) from rdb$fields as t1,rdb$types as t2,rdb$collations as t3,rdb$functions as t4) and ((("atlz" = "atlz	SQL Injection

Predicted vulnerabilities are shown in the second column of the table, whereas SQL queries, XSS, and RFI coding instructions are shown in the first column.

Conclusion
The results of our comprehensive testing of function vulnerability categorization using n-grams, suffix trees, and trivial characteristics allow us to make numerous conclusions. First, using the 74% accuracy we achieved from "character diversity "as a baseline criterion, it is clear that extracting multiple n-grams does not seem to provide strong classification results at this time. We also found that the overall outcome remained unchanged even when n-gram combinations were manually picked (in a way that would often be considered unlawful and result in over fitting). But this analysis proves a key point: even seemingly insignificant traits may reveal a lot about a function's vulnerability. To make the outcomes even better, this study may be done in a few different ways. To start, maybe we can come up with some more insignificant aspects to look at. The second thing to consider is trying out other n-gram selection methods and other classification parameters (beyond the defaults) in the Sci Kit library. Third, instead than focusing on "character diversity," it might be instructive to examine which strings of characters are most crucial. One approach would be to do the character variety tests again after pre-processing removes certain strings, such as rectangular brackets, curving brackets, ++, etc. At last, it's feasible to see whether the methods discussed in this article work for other languages than C when it comes to

finding security flaws.

References

1. Enron email dataset. Available from: https://www.cs.cmu.edu/~enron/. Accessed: 2017-07-01.
2. National vulnerability database. Available from: https://nvd.nist.gov. Accessed: 2017-07-01.
3. Basile VR, Briand LC, Melo WL. A validation of object-oriented design metrics as quality indicators. IEEE Trans Software Eng. 1996;22(10):751-761.
4. Brumley D, Cheoah T-c, Johnson R, Lin H, Song D. Rich: Automatically protecting against integer-based vulnerabilities. Department of Electrical and Computing Engineering; c2007. p. 28.
5. Door N, Rodhe M, Saga M. CSSs': Towards a realistic tool for statically detecting all buffer overflows in C. In: ACM Subplan Notices. 2003;38:155-167.
6. Evans D, Larochelle D. Improving security using extensible lightweight static analysis. IEEE Softw. 2002;19(1):42-51.
7. Goldenrod P, Levin MY, Molnar D. Sage: Whitebox fuzzing for security testing. Queue. 2012;10(1):20.
8. Haller I, Lewinski A, Neguswanderer M, Bos H. Dowsing for overflows: A guided fuzzer to find buffer boundary violations. In: USENIX Security Symposium; c2013. p. 49-64.
9. Hassan AE. Predicting faults using the complexity of code changes. In: Proceedings of the 31st International Conference on Software Engineering; c2009. p. 78-88.
10. Kim S, Zimmermann T, Whitehead EJ Jr, Zeller A. Predicting faults from cached history. In: Proceedings of the 29th International Conference on Software Engineering; c2007. p. 489-98.
11. Larochelle D, Evans D, *et al*. Statically detecting likely buffer overflow vulnerabilities. In: USENIX Security Symposium; c2001. p. 32.
12. Lather P, Shah R, Srinivasa K. Stacy-static code analysis for enhanced vulnerability detection. Cogent Eng. 2017;4(1):1335470.
13. Ma R, Yan Y, Wang L, Hu C, Xu J. Static buffer overflow detection for C/C++ source code based on abstract syntax tree. J Residuals Sci Technol. 2016;13(6).
14. Moser R, Percy W, Sulci G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proceedings of the 30th International Conference on Software Engineering; c2008. p. 181-90.
15. Naprapathy RM, Mirin BG, Levine M. A suffix tree approach to antispam email filtering. Mach Learn. 2006;65(1):309-338.
16. Pantile E, *et al*. Improving C++ software quality with static code analysis. N/A; c2014.
17. Sachem H. The geometry of innocent flesh on the bone: Return-into-lab without function calls (on the x86). In: Proceedings of the 14th ACM Conference on Computer and Communications Security; c2007. p. 552-61.
18. Shin Y, Williams L. An empirical model to predict security vulnerabilities using code complexity metrics. In: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement; c2008. p. 315-17.