My research interest lies in solving security and software engineering problems via program analysis, with the focus on challenges caused by the opaqueness in modern complex computing systems, which may consist of various operating systems, virtual machines, containers, data processing procedures and artificial intelligence (AI) models. Due to their complexity, it is highly challenging to understand their internal workings. Such opaqueness is the root cause of many security and software engineering problems. I have developed novel techniques to make these systems transparent through the unique program analysis perspective. The key observation is that the various components in such systems are essentially programs. For example, OS can be considered a gigantic event-driven program and an AI model is usually a Python program dominated by matrix computation. My techniques fall into three categories: system operation transparency, data processing transparency and deep learning transparency.

**System operation transparency** is critical for system security and problem diagnosis. For example, advanced persistent threats (APTs) are highly sophisticated, multi-staged attacks that act slowly and deliberately over a long period of time to expand their presence in an enterprise environment and reach their goals. However, the opaqueness of enterprise systems makes it difficult to infer causality between operations (e.g., sensitive files being stolen and visiting a malicious website a few weeks ago) so that attacks cannot be understood and defended. I have developed novel techniques [4, 9, 12, 13, 14, 15, 18, 19] that log critical system/application events that are semantically rich to provide system level transparency. My research outcomes have been published on USENIX Security, NDSS, USENIX ATC, ACSAC and so on, and received the Distinguished Paper Awards from the USENIX Security Symposium 2017 and ISOC NDSS 2016. My techniques reduce the run time overhead of logging system operations from over 40% to less than 7% and space overhead from 155.5 GB for 3 months to 1.6 GB. My techniques have been extensively evaluated in the DARPA Transparent Computing program and the ONR Learn-2-Reason program and produced good results. Some of them [12] are in the process of being integrated to Redhat Linux and some [15] were shipped to the Sandia National Laboratories. My tools [13, 15] are also being integrated to the Navy Trident Warrior training program aiming to train Navy soldiers to defend cyber attacks.

**Data processing transparency** is critical for the trustworthy of processing results. Many security applications such as spam filtering and social network malicious community (e.g., zombie fans and spam accounts) analysis heavily rely on data processing. Defects in data processing procedures could lead to security breaches and bogus findings that could lead to waste of resources. I developed novel techniques to provide transparency to data processing by on-the-fly partial derivative computation and probabilistic inference. Results were published on top-tier conferences such as ICSE and FSE [8, 20]. My techniques successfully identified defects in six existing spam filters and improved accuracy from 50%-70% to over 96%.

**Deep learning transparency** is critical for deep neuron network (DNN) security (e.g., detecting adversarial samples) and accuracy. Without understanding DNN internals, it is hard to understand the causes of adversarial samples and low model accuracy. DNN models are essentially programs consisting of a sequence of matrix computations and activation functions. With the support of my advisors, I initiated and supervised a group of four other Ph.D. students to address the deep learning transparency problem from the program analysis perspective, and we have achieved many state-of-the-art results [7, 10, 11, 17]. In the security domain, my technique [11] (NDSS’19) uses DNN invariants (intuitively, neuron activation patterns of benign inputs) for adversarial sample detection and achieves over 96% detection accuracy on 11 attacks and 13 different models, outperforming many other state-of-the-art techniques, and our model hardening technique [17] (NeurIPS’18 Spotlight) rejects inputs if the model does not make predication based on appropriate features (e.g., human eyes/noses/mouths in the face recognition task) and is able to filter out 94% adversarial samples for 7 different tested attacks. In the model accuracy domain, my technique [10] (FSE’18) performs differential analysis to identify the root cause of low test accuracy and input selection to resolve the problem. Our evaluation shows that this method improves model accuracy from 75% to 93% on average for 20 models whereas the state-of-the-art can only improve to 85% with 11 times longer training time. These techniques have attracted world wide collaborations with PSU, Univ. of Minnesota-Twin Cities, HKUST, HKU (Hong Kong), NJU (China) as well as the X-Lab which is an industrial research lab working on the Apollo auto-driving systems [2] supported by Baidu. A number of our ideas and developed tools are being transferred to the X-Lab projects (e.g., Apollo), and I even raised $30,000 fund from them for my advisor.

**System Operation Transparency**

Recent cyber attacks, especially APT attacks are becoming increasingly sophisticated and targeted, leading to significant damages such as data leak (e.g., huge swaths of confidential documents were stolen in the Sony Picture attack), financial loss (e.g., the Carbanak attack causes over $1 billion loss) and even threatening human life (e.g., the Stuxnet attack targets nuclear power equipments). OS level provenance tracking is a very important
approach for APT attack investigation. Starting from the symptom events (e.g., the creation of a malicious process), investigators perform backward or forward tracking on the provenance data based on dependency relations (e.g., a process reads/writes a file). Traditional techniques suffer from two problems: low analysis accuracy, and significant run time and space overhead. My techniques try to solve these two problems.

**Accurate dependency analysis:** The low analysis accuracy problem in traditional techniques is mainly caused by the dependence explosion problem: for long running processes, a subject (i.e., process) is causally related to all the objects it has accessed so far, which introduces many false dependencies. Prior to my work, the state-of-the-art work BEEP [5] partitions these long running executions to units based on individual iterations of event handling loops which represent a common design pattern in these programs. An event handling loop accepts external requests as inputs and then dispatches them to different functions for processing in individual iteration. Each iteration constitutes a unit. With the partitioning, events in different execution units are considered independent, which helps remove many false dependencies. However, BEEP has many limitations. Firstly, execution units in BEEP expose low level semantics, which are hard for investigators to understand. Secondly, BEEP generates excessive units (e.g., units for GUI events like key strokes or mouse movements). Thirdly, to find dependency relations across units, it requires a heavy-weight training process. Moreover, it performs binary rewriting to instrument the partitioning logic, which is not practical for complex programs like Firefox. To solve these problems, I proposed MPI [13], a multiple perspective attack investigation technique with semantics aware execution partitioning. Users first annotate the target program to define desired investigation perspectives with the assistance of our annotation helper tool. These perspectives reflect high level semantic partitioning of the target program, such as individual tabs or individual web pages for Firefox. MPI then automatically analyzes and instruments the target program to produce provenance aware applications that can proactively report its current context (e.g., current tab ID in Firefox) to the OS provenance collection system, such that events in different contexts are independent. Doing so, MPI provides accurate, multiple perspective and semantics rich attack investigations with less instrumentation and run time overhead. MPI received the Distinguished Paper Award at the USENIX Security Symposium 2017.

**Effective and Efficient Provenance Collection System Design:** Prior to my work, widely used provenance collection systems like the Linux Audit framework have substantial run time and storage overhead. According to our profiling results [12], the root cause is the large amount of redundant events which require excessive computing resources to transfer, log and store. To address these problems, I proposed ProTracer [14], a lightweight provenance system that features online redundancy reduction. It uses memory ring buffers as the data channel to speedup the data transfer process, and a novel online log redundancy detection and reduction algorithm to reduce both the run time overhead and space overhead via alternating between tainting system objects/subjects and logging system events. It treats all incoming inputs to the system as taints, and propagates them through dependency relations. Following the principle that persistent system events (e.g., process creation) should be logged and temporary system events (e.g., duplicated file reads) are used to propagate taints, we carefully designed a set of logging and tainting rules to detect and reduce redundant events without affecting the completeness and correctness of dependency analysis. The evaluation with different realistic system workloads and a number of attack cases show that ProTracer only consumes 1.28% of the space used by state-of-the-art provenance systems, 7 times smaller than a recent off-line garbage collection technique with less than 7% run time overhead for servers and less than 5% run time overhead for regular applications. Moreover, the investigation process is also 7 times faster using reduced log files. ProTracer received the Distinguished Paper Award at Internet Society Network and Distributed System Symposium 2016.

**Other Related Projects:** Some of my other works also fall into this domain including instrumentation free accurate causality analysis on Windows [9], model based causality inference [4], kernel level log reduction [12], machine learning based provenance analysis [15], hypervisor level provenance analysis [18] and shared library level support for provenance analysis [19]. These papers are published on NDSS, USENIX ATC, ACSAC etc. Moreover, a number of our tools are reused by other researchers from UIUC, SRI International and so on, and have been shipped to Sandia national Laboratories, DARPA, ONR and Navy.

**Data Processing Transparency**

Many real world applications are powered by data processing components. For example, spam filters usually use the Bayesian classifier or its variants. Many malicious community analyses (e.g., identifying zombie fans, spam social accounts, and phishing websites), social network analyses (e.g., finding social hubs and ranking users), and even malware analyses (e.g., detecting packers at the binary level) heavily depend on graph based algorithms like node clustering or node ranking. The results of these systems are usually hard to understand due to the opaqueness. Most existing techniques try to interpret them by tracking program dependencies. Namely, for a given output, they try to identify the set of inputs that affect the output value by analyzing program dependencies.
Despite that the method works well on traditional program analysis tasks, it is not suitable for data processing systems. For example, the widely used node ranking algorithm PageRank iteratively updates all node values based on the information from their neighbors. Due to the rippling effects, all nodes will eventually transitorily depend on the rest of the whole graph, leading to massive dependency relations. To solve this problem, I proposed LAMP [8] which introduced the concept of weighted dependency analysis for graph based algorithms to highlight important connections or nodes that play important roles for a final output value. To measure the importance of different dependencies, we use partial derivatives of a given node with respect to all nodes it depends on.

I developed a technique that is similar to automatic differentiation to calculate partial derivatives. In addition, it allows derivative-like computation on discrete behaviors such as control flow selections by forking parallel executions and computing the partial “derivatives” by applying tiny perturbations. Our evaluation and case studies have shown the usefulness of LAMP on many important tasks such as improving the accuracy of spam filters from 50% to 70% to over 96% for six existing models, explaining the results of social network analysis tasks and helping bug fixes in Python based graph model implementations.

Deep Learning Transparency

Detecting Adversarial Samples: DNNs are vulnerable to adversarial samples that are generated by perturbing correctly classified inputs to cause DNN models to misbehave (e.g., misclassification). This can lead to disastrous consequences in security sensitive applications. Existing defense and detection techniques work well for specific attacks [11] but do not generalize well on different types of attacks. The execution of a DNN model is just like a program execution consisting of a sequence of matrix computations and activation functions. In the traditional software security domain, researchers proposed program invariants checking to limit the program behaviors to benign ones. For example, control flow invariants are used to check control flow integrity, and a violation indicates an attack. Inspired by this, I proposed DNN invariants to detect adversarial samples. Similar to program invariants, DNN invariants represent benign behaviors of a given model. Unlike traditional program invariants, which are logical constraints, DNN invariants are probabilistic models. There are two types of DNN invariants, the value invariant and the provenance invariant. A value invariant is activation value distributions of a specific layer for benign inputs, and a provenance invariant is activation transition patterns of two consecutive layers for benign inputs. Together, value and provenance invariants model the benign behaviors of a DNN model. A violation of any invariant indicates an adversarial sample input. We empirically compared our technique, NIC, with four other state-of-the-art detection methods (i.e., LID, MagNet, HGD, and Feature Squeezzing), and found that NIC is more general to different types of attacks with over 96% detection accuracy on 11 different attacks and 13 different models. This work will appear in NDSS 2019. Currently, it is being transferred to the X-Lab and applied to detecting adversarial samples in auto-driving systems.

Improving Model Accuracy: Training DNN models is not easy, and many factors can result in low accuracy models. For example, biased training inputs can lead to over-fitting (when training inputs are too similar to each other) or under-fitting (when training inputs are bad samples that do not represent the general features shared by other real world data points) models with low accuracy. Prior to my work, state-of-the-art techniques use various data augmentation methods such as predefined domain specific input augmentation operations (e.g., mirroring and rotation for image inputs) or generative models (e.g., generative adversarial networks) to generate new inputs, and retrain the model with the new inputs. We studied these techniques and found that they are inefficient and ineffective with no guarantees in improving the over-fitted or under-fitted models. This is because they lack the understanding of the root cause of the low model accuracy. To solve this problem, I proposed MODE [10] (FSE’18), a differential analysis and input selection technique to improve the over-fitted and under-fitted models. MODE first creates heat-maps which quantitatively measure the importance of individual neurons by using the SoftMax function which is a generalized logistic regression function. Then it generates differential heat-maps to find the neurons with significant diverse importance for correctly classified cases and mis-classified cases by comparing heat-maps generated from the two types of inputs. These neurons are highly associated with the low model accuracy. After that, MODE selects new training inputs that can potentially reduce such discrepancy and retrains the model. The experimental results on 40 over-fitted and under-fitted models show that MODE can improve the model accuracy from around 75% to over 93% while the state-of-the-art can only improve to 85% with 11 times more training time. MODE is currently being used by other researchers from NUS (Singapore), NJU (China) and also the X-Lab.

Other Projects

Besides these, I also participated in many other projects such as high availability virtualization systems [3] (COLO has been admitted to the Xen code base), mobile security [16], fuzzing [21, 22] and program synthesis [23].
Ongoing and Future Works

I envision that intelligent systems, which combine the traditional computing components with AI models, will be increasingly popular. For example, in most auto-driving cars, sensors capture the environment status and transfer the data to AI models for tasks like object detection. Based on predictions of AI models, the traditional computing components including the route planning and physical control algorithms will take over and perform the corresponding actions. In addition, popular digital assistant programs (e.g., Google Assistant and Siri) use AI models for perception (e.g., recognizing voice inputs) and then perform concrete tasks with traditional computing components (e.g., taking pictures). I plan to study such intelligent systems from three perspectives: security, dependability and productivity.

Intelligent System Security

I would like to apply my security knowledge to the intelligent system domain, with the focus on three important problems: vulnerability finding, system hardening and forensics analysis.

Vulnerability finding: Intelligence systems introduce new interactions between traditional computing components and AI models. As a result, inputs to the traditional component will affect the behavior of AI models, and vice versa, which brings more challenges in vulnerability finding as the required analysis has to go across two sides. For example, in the Apollo auto-driving platform, images are taken by cameras, preprocessed by a few traditional computer vision algorithms and then sent to the object detection models. Perturbing physical objects (e.g., adding sticky notes to stop signs) can fool the AI models. To understand this phenomena, the preprocessing algorithms have to be analyzed besides the AI models. On the other hand, prediction results from the AI models will be used for route planning and car control which are powered by traditional computing. Thus, errors from AI models may significantly affect car behaviors. In the reported Tesla auto-driving car accidents [1], we can find that most accidents involve the participation of both sides. Fuzzing, which tests a target program by generating new inputs via mutating given seed inputs, is commonly used in vulnerability finding for traditional computing components. Other techniques like symbolic executions also try to expose possible behaviors of the target program by exploring the input space. In AI model vulnerability finding, a commonly used technique is to add small perturbations to existing inputs to test AI models. As such, I plan to create a new uniform vulnerability finding platform which can support the analysis of both sides.

System hardening: AI models are vulnerable to attacks such as adversarial samples and parameter stealing, which calls for AI model hardening techniques. Traditional computing components can be hardened by logical assertions which limit the program behaviors only to the allowed ones. For example, assertions limiting control flow transitions to benign targets can help ensure control flow integrity, and assertions checking the run time values of certain variables can help prevent format string attacks. Inspired by this, I plan to harden AI models by enforcing them to respect certain logical constraints which essentially are properties of AI model benign behaviors. For example, in our previous work, AmI [17] (NeurIPS’18 Spotlight), we proposed a novel technique to assert the logical constraint that model predictions must be made based on human understandable features in the DNN, and applied it to defend adversarial samples in face recognition systems. The key technique is to find one-to-one mappings between human understandable features and neuron activation patterns so that the logical relations can be asserted. Our results show that the enhanced model can filter out over 94% adversarial samples generated by 7 different attacks. This shows the effectiveness of regulating AI model behaviors by logic assertions, and I plan to extend this work to more intelligent systems besides face recognition with more diverse logical assertions.

Forensics analysis: Forensics analysis for intelligent systems requires the dependency tracking capability for both traditional computing and AI models. For the traditional computing side, there are many existing provenance systems (e.g., ProTracer) and analysis methods (e.g., backtracking), while for the AI model side and the interaction of the two, there is limited support. In AI models, the output usually depends on all the weights in the model and the whole input, which makes it hard to analyze. I plan to extend the quantitative dependency importance analysis idea proposed in LAMP to this domain so that we can perform probabilistic forensics analysis across both sides. In addition, provenance tracking for intelligent systems also have to be effective, i.e., providing semantics rich data to support the analysis, and cost effective, i.e., causing as little run time and storage overhead as possible. I plan to extend my work to effective and efficient provenance collection and analysis in intelligent systems.

Intelligent System Dependability

Just like traditional computing components inevitably contain bugs, AI models may have undesirable behaviors (referred to as model bugs), which will affect their dependability. Different from traditional coding bugs, model bugs are misconducts in the model engineering process. For example, biased training inputs can lead to various consequences such as over-fitting or under-fitting, stereotyping defects (i.e., undesirable features are picked up and even amplified by the model to make mis-prediction) and robustness defects. The quality of input embedding
in natural language processing tasks (i.e., translating words or sentences to numerical representations) will significantly affect the performance of AI models. Inappropriate reward value settings in reinforcement learning will make the training phase unexpectedly long or even failing to converge. Similar problems can also happen in other types of AI models if the hyper-parameters (e.g., the learning rate) are not properly set. In traditional software engineering, testing and debugging are essential steps to locating and fixing bugs. Similarly, I believe that testing and debugging will be necessary steps in the intelligent system development. I have started to explore possible testing and debugging techniques in the intelligent system domain (e.g., my work MODE on FSE’18 [10]), and attracted collaborators from Univ. of Minnesota-Twin Cities and HKUST.

Intelligent System Productivity

Many traditional computing tasks are parameterized (i.e., the quality of results depends on tunable parameters) or environment dependent. For example, commonly used image sharpening algorithms require three or even more tunable parameters, and these parameters have to be adjusted for individual images to achieve the best result. A widely used robotic vehicle control software Ardupilot has over 600 tunable parameters. A data center cooling control system is highly environment sensitive. It has to monitor the data center environment and the cooling system status to plan a good schedule. These programs require a lot of human efforts or are based on simple (and hence likely ineffective) heuristics to operate properly. Intelligent systems with AI models can improve this situation. For example, trained model can be used in data center cooling control to predict parameter values based on the environment, such as temperatures and the calendar (e.g., weekends and holidays). Some recent techniques tried to use intelligent systems for such tasks. However, they still require a lot of human efforts to identify important features as inputs to AI models, construct training data (e.g., labeling individual operations in supervised learning), setting hyper-parameters (e.g., rewards in reinforcement learning), design the model, and conduct the training. I envision that this process can be highly or even fully automated with program analysis techniques. For example, program analysis can be used to identify program variables that denote important features. We have made our first step in this direction and the paper was accepted to CGO 2019 [6]. I will continue the exploration.

References

