Research Statement

Wei Yang

I enjoy doing research in Computer Security and Software Engineering and specifically in mobile security and adversarial machine learning. A primary goal of my research is to build adversarial-resilient intelligent security systems. I have been developing such security systems for the mobile device ecosystem that serves billions of users, millions of apps, and hundreds of thousands of app developers. For an ecosystem of this magnitude, manual inspection or rule-based security systems are costly and error-prone. There is a strong need for intelligent security systems that can learn from experiences, solve problems, and use knowledge to adapt to new situations.

However, achieving intelligence in security systems is challenging. In the cat-and-mouse game between security analysts and adversaries, the intelligence of adversaries also increases. In this never-ending game, the adversaries continuously evolve their attacks to be specifically adversarial to newly proposed intelligent security techniques. To address this challenge, I have been pursuing two lines of research: (1) enhancing intelligence of existing security systems to automate the security-decision making by techniques such as program analysis [11, 8, 10, 6, U6], natural language processing (NLP) [9, 7, U7, 1], and machine learning [8, 4, 3, 2]; (2) guarding against emerging attacks specifically adversarial to these newly-proposed intelligent security techniques by developing corresponding defenses [13, U1, U2] and testing methodologies [12, 5].

Throughout these research efforts, my general research methodology is to extract insightful data for security systems (through program analysis and NLP techniques), to enable intelligent decision making in security systems (through machine learning techniques to learn from the extracted data), and to strengthen robustness of the security systems by generating adversarial-testing inputs to check these intelligent security techniques and building defense to prevent the adversarial attacks.

With this methodology, my research has derived solutions that have high impact on real-world systems. For instance, my work on analysis and testing of mobile applications (apps) [11, 10] in collaboration with Tencent Ltd. has been deployed and adopted in daily testing of a mobile app named WeChat, a popular messenger app with over 900 million monthly active users. A number of tools grown out of my research have been adopted by companies such as Fujitsu [P1, P2, 13, 6], Samsung [12, 5], and IBM.

1 Enhancing Intelligence of Security Systems

The wide adoption of personal digital devices such as mobile phones increases the need of security systems that can serve for users without information technology expertise. Security systems need to evolve from just warning expert users about potential security threats to making security decisions for common users. Thus security systems are required not only to capture security-sensitive behaviors but also to infer the intentions of security-sensitive behaviors. For example, a benign app may send out a user’s location to find nearby restaurants, while a malicious app may leak the user’s location for its own benefits.

To address this issue, my dissertation research takes a different approach from previous security systems: enhance the intelligence of security systems by mimicking human decision-making process. I have designed and implemented security systems based on a key factor that is frequently overlooked by existing security systems: user expectations, i.e., did a user expect a certain functionality (e.g., sending user’s location) to occur? My research contributions range from automatic assessment of security risk [7] and privacy risk [9], through automatic synthesis of natural language security descriptions [U7], to automated malware detection [8, 13].

Of my research contributions, one highlight is AppContext [8]. On mobile app markets, I find a large amount of evasive malware that hides malicious intentions by mixing malicious behaviors with expected functionality. For example, a malicious app may present itself as a messaging app that sends SMS messages when the user clicks the send button. However, it also sends SMS messages containing the user’s contact information in the background without notifying the user. Since both behaviors use the same set of security-sensitive permissions and APIs, existing security systems such as information flow analysis are unlikely to distinguish between these cases.

To address this issue, I design and build AppContext by considering user awareness of the security-sensitive behaviors. AppContext reveals a key insight that mobile malware leverages two unique characteristics of mobile systems to maximize profits while prolonging its lifetime by deceiving users: frequent occurrences of imperceptible system event and indicative states of external environments. Following such insight, I develop static analysis in AppContext that analyzes executable code of a mobile app to extract the contexts of security-sensitive behaviors that precisely reflect user awareness and the intention of the behaviors. These contexts include (1) system events that malware uses to frequently trigger malicious payloads and (2) external-environment states that malware uses to control the payloads not to trigger too frequently for users to notice anomalies. For example, a malicious app will trigger its payload whenever phone signal strength changes but only during 11PM to 5AM when users are usually sleeping. AppContext then trains a machine learning model based on the contexts to differentiate benign behaviors from malicious ones. Our evaluation results indicate that AppContext can detect malware (including unknown malware) with 87.7% precision and 95% recall.

Wei Yang, Research Statement, p. 1 of 5
In addition to bringing the concept of user expectation in program analysis, I also work on various types of textual artifacts to infer user expectations. WHYPER [7] is our pioneering and exemplary work in this space, which automates the risk assessment of Android apps by applying NLP techniques to app descriptions. App markets such as Google Play present a permission list to show what private data an app may access, rather than how and why the app uses the private data, causing users to make uninformed decisions on how to control their privacy. To address this issue, my collaborators and I develop an NLP technique based on semantic models extracted from Android API documents to determine which sentence (if any) in app descriptions indicates the use of a permission. WHYPER was the first to apply NLP to the mobile security domain to analyze the fidelity between app descriptions and permissions. Our results on 581 popular apps show that WHYPER effectively identified the permission-explaining sentences with 82.8% precision and 81.5% recall.

Following WHYPER, my collaborators and I develop a number of approaches that further bridge the semantic gap between user expectations and apps’ security-sensitive behaviors. The most closely related ones are CLAP [U7] and Pluto [9]. CLAP [U7] naturally extends WHYPER by automatically generating meaningful explanations for unexplained permission uses. Prior approaches such as WHYPER rely on the availability of permission-explaining sentences in the app description. If an app does not provide description or provide uninformative description, none of the previous approach would work. To address this issue, we borrow the idea of collaborative filtering in recommendation systems (e.g., recommending movies to one user with the help of other users’ ratings). CLAP uses information retrieval and text mining algorithms to identify explanatory sentences in similar apps and use them to synthesize sentences that can explain permissions for the original app. Our results show that CLAP is effective for generating highly interpretable natural language explanations for unexplained permission requests with over 90% precision.

Pluto [9] takes a step further from WHYPER to assess and quantify the risk of potential exposures of user data (that the users expect to be secure to provide to apps). Pluto leverages NLP, machine learning, and data mining techniques to reveal what private information can potentially be inferred from user inputs, files, and the names of apps installed on the phone. To validate Pluto, we conduct a user study, which establishes the ground truth for user input data and lists of installed apps for about 300 users. Our results show that Pluto achieve 75% recall and 80% precision for user data from app files and user inputs, and even better results for the names of installed apps.

2 Strengthening Intelligent Security Techniques

Although the above-mentioned intelligent techniques bring impressive capabilities to security systems, the robustness of these techniques in adversarial settings is still questionable. My research has explored the feasibility of developing adversarial-resilient techniques in two main areas: program analysis and machine learning.

Adversarial-resilient program analysis. To evade the detection of security systems, adversaries may obfuscate the malicious program code or change the malicious program structures to impede or misguide program analysis. To build more robust security systems, I design and implement program analysis techniques [13, U1, U2] that are resilient to obfuscation and evasive attacks. One of such techniques is EnMobile [13], a program analysis framework characterizing mobile-app behaviors by directly and comprehensively modeling an app’s interactions with its environment. When developing AppContext, I observe that many evasive malware samples separate malicious behaviors in multiple phases (e.g., downloading, preprocessing), with intermediate computation results stored in temporary files or databases. Existing information flow cannot “stitch together” the segmented flows punctuated with interactions with external entities (e.g., files or databases) to decipher malicious behaviors initiated and controlled by malicious servers, such as initiating spams or launching denial-of-service attacks.

To address this challenge, I propose the concept of entity-based program analysis to complement traditional program analysis based on implementation-dependent structures (e.g., methods, objects). To enable entity-based program analysis, I design two supporting components: an identity-propagation component that conducts a flow- and context-sensitive analysis to establish the correspondence between in-program objects and the external entities with which the object may interact in each execution context, and a stitching component that conducts a flow-sensitive analysis to connect segmented information flows that are feasible in actual executions. I implement EnMobile and provide a practical application of EnMobile in a signature-based scheme for detecting mobile malware. Our evaluation results on a set of 6,614 apps show that EnMobile detects malware with substantially higher precision and recall than state-of-the-art approaches. While developing EnMobile, I observe that malware further evolves to use program features that cannot be analyzed by static analysis such as native code, dynamically loaded code, or dynamic programming language features (e.g., Java reflection). I address this issue from two different perspectives. On the one hand, I develop ModuDroid [U1] to enable partial installations. ModuDroid generates patches to separate the suspicious code (e.g., code unanalyzable to static analysis) from desired code (e.g., code passed the static checking) in an Android app. ModuDroid guarantees the separation to be impact-free at the component level and our evaluation on 968 benign apps and 977 potentially unwanted apps shows that only fewer than 5% of separated apps can be unsafe. Our evaluation results also show that ModuDroid can successfully separate more than twice of the apps than related approaches.

On the other hand, I propose REINAM [U2] to model the unanalyzable code by automatically inferring grammars...
of the inputs that can be accepted by the unanalyzable program parts. Specifically, REINAM infers the input grammars based on observations from executions of the code. REINAM leverages reinforcement learning to diversify the seed inputs so the dynamic executions will not narrowly focus on a certain area of a program. Our preliminary results suggest that the grammars generated by REINAM are at least four times more comprehensive (while maintaining the same precision) than grammars inferred by existing techniques such as active learning techniques.

**Adversarial testing for machine learning.** Recent research finds that machine learning algorithms can produce unexpected results to small, specially crafted perturbations. Such perturbations cause learning-based systems to misclassify these well-crafted examples. In security systems, such incorrect behaviors can lead to potentially disastrous consequences. Traditional testing system is not suitable for detecting these incorrect behaviors because the core logics of learning-based systems are embedded in the machine learning models (i.e., arithmetic operations of formulas) instead of the control flow program structures (i.e., program branches/paths) that traditional testing system is based on. An automated testing framework is needed to enable a learning-based security system to detect erroneous behaviors and correct the behaviors before adversaries launch attacks.

MRV [12] is the first of such system that generates adversarial-testing inputs for mobile malware detection systems. Existing approaches on adversarial input generation typically focus on image inputs. Some prior work applies its attack on malware but only manipulates the feature vectors of a malware sample without considering feasibility and impact of the mutation on the malware’s code. Based on our study, when applying the changes of prior work made in feature vectors to the malware’s code, the changes cause the malware to crash, cause undesired behaviors, or disable malicious functionalities (sometimes the modified code cannot even be compiled). To address these issues, I design and implement a systematic adversarial-testing input generation system, Malware Recomposition Variation (MRV).

The test inputs generated by MRV satisfy three requirements: malicious (i.e., the generated malware maintains the original malicious purposes), robust (i.e., the generated malware does not crash), and adversarial (i.e., the generated malware can evade the detection of security systems under testing).

To generate such test inputs, I design two input generation strategies (i.e., feature confusion and feature evolution) that follow structures of existing evasive malware or existing malware evolution histories. Upon the given malware, MRV conducts semantic-feature mutation analysis and phylogenetic analysis to synthesize mutation strategies. I build a program transplantation framework capable of inter-method, inter-component, and inter-app transplantation to automatically mutate malware bytecode based on synthesized mutation strategies to generate new malware variants (i.e., test inputs). Our evaluation on existing research and commercial malware detectors shows that MRV is effective to generate adversarial-testing inputs that can evaluate the differentiability of selected features and the robustness of a malware detection model. For these malware detectors, MRV produces 5 to 70 times more adversarial-testing inputs compared to existing adversarial learning approaches.

## 3 Future Directions

In general, I plan to continue and expand my research in software engineering and security, devising techniques that make security systems more intelligent and robust. I believe that the intersection between program analysis, machine learning, natural language processing, and security is a rich area for future research. New technologies will benefit from interdisciplinary research and advance our understanding of design principles of intelligent security systems.

Specifically, I am enthusiastic to continue my research in two main lines of future work.

**Intelligent security techniques with little labeled data.** We are living in a world that has many orders of magnitude more data than labels. Most of existing intelligent security techniques including those that I have developed are based on labeled data. Such factor poses barriers for adopting these existing intelligent security techniques in practice because the evasive nature of security subjects such as malware would make the labeling process erroneous and laborious. In the next few years, I plan to address the problem of lacking labeled data from three directions.

First, in the short term, I plan to leverage multiple sources of data to complement each other when labels in one or more data sources are missing. My existing work already extracts program behaviors from different perspectives by analyzing multiple types of artifacts, including static code information, dynamic execution information, textual information such as app descriptions and API documents, and graphical information such as user interfaces. My next goal is to address security-decision making in such security systems using techniques of multimodal representation learning. My existing work already builds modality-specific approaches for each data modality (i.e., each type of data). Based on these modality-specific approaches, I plan to learn joint representations that are shared across multiple modalities. The first essential step is to develop techniques of customized graph embedding to cover program structures such as control-flow graphs or call graphs into high-dimensional vectors.

Second, in the mid term, I plan to tackle the problem by considering the inverse problem of analyzing data: generating data. Many fields have achieved a certain level of success in generating data (e.g., generating images, synthesizing small programs) by using generative models, which do not require any labeled data. I believe that an even more practical use of generative model is to analyze data by learning to generate the data. Through learning to generate a type of data, generative models learn to understand the properties of the data. For example, by generating evasive malware, a generative model forms a relatively complete notion of what makes a malware evasive. Missing
any important characteristic of a malware would make the model generate non-evasive malware or even non-malicious apps. An analogy is that in a generative adversarial network (GAN), the existing research typically is to improve the performance of the generator in a GAN, whereas our focus here is to improve the classifier while the generator becomes more effective.

Last, in the long term, I plan to explore the possibility of using transfer learning to store knowledge gained in solving the tasks with labeled data and apply such knowledge to solve problems where labeled data are not available. For example, the data of vulnerabilities and attacks on mobile apps are abundant, while such information is lacking on Internet of Things (IoT) apps. I plan to investigate ways to transfer the knowledge learned from mobile apps and apply such knowledge on IoT apps. We have already made a first step in such direction by transferring knowledge about inter-component vulnerabilities in Android apps towards inter-rule vulnerabilities in IoT apps [U4].

Testing (deep) learning-based security systems. A natural extension of my current research is to test (deep) learning-based security systems. I plan to advance this direction from three different perspectives.

First, in the short term, I plan to continue working on adversarial testing of learning-based security systems. In addition to generating adversarial-testing inputs, I plan to exploit other potential vulnerabilities such as the privacy vulnerabilities in learning-based security systems. In one of my undergoing projects, we develop a metamodel [U3] that can infer the private properties of the training data of certain black-box machine learning models. Although such issue is not unique to security systems, the consequences of exposing these private properties of the security data would be disastrous. For example, for an anti-virus system, if the configuration of the sandboxing can be inferred from the model and the training data, adversaries can devise malware to specifically evade the detection of the sandboxing. I would like to investigate the potential exposures of such private properties in the testing phase of security systems and develop techniques to mitigate such potential exposures.

Second, in the mid term, I plan to propose and evaluate new testing metrics for learning-based security systems. Traditional testing metrics (e.g., statement, branch, and path coverage) do not work when testing learning-based systems because core logics of learning-based systems are embedded in the machine learning models (i.e., arithmetic operations of formulas) instead of control flow statements in traditional software. Prior work proposes neuron coverage (i.e., the ratio of the number of distinct activated neurons to the total number of neurons in the neural network) as the testing metric. There is one key limitation in this testing metric: it interprets neuron activation as a positive indicator for testing effectiveness, while it is only a status of the respective neuron. I plan to propose combinations of neuron activations and manifolds of training data as testing metrics [5]. Lower-dimension manifolds are good models for many data-related tasks, whose data points might lie in very high-dimensional spaces. Programs can reject inputs that are far away from natural manifolds because these inputs are well beyond the confident regions of the ML model under testing. Such factor also indicates that manifold can tell whether testing inputs are meaningful to the ML model under testing (i.e., reaching the deeper logics of the model).

Last, in the long term, I plan to generate training/testing environments identical to the real-world environments for lab training/testing of safety- and security-critical systems. The training and testing of such systems usually involve techniques (such as reinforcement learning) that need to experience a huge number of failures before a correct model is learned. It is too expensive to develop such model directly in the real-world environments. I plan to develop a generative technique that can generate a model of environments capable of predicting several steps into the future (e.g., how other cars would respond to a stop sign). With this model, I can then perform fuzz testing to mutate various elements in the environments to see how the system under testing responds.

4 References

Refereed Conference Papers


Refereed Journal Articles & Workshops


Under Review / In Preparation


*The first two authors contributed equally to this work


*The first two authors contributed equally to this work


Patent
