# RESEARCH STATEMENT

Yuchen Zhou (yuchen@virginia.edu)

Modern applications increasingly rely on code and services from multiple parties. The need for extra functionality and better modularity drives software developers to employ third-party solutions. These solutions differ from the traditional libraries (such as libc) since execution of integrated third-party code relies on communications with their private back end server. The recent emergence of social network giants such as Facebook and Twitter, as well as multiple advertising and analytics services have made almost every mobile or web application a complicated mash-up. A recent study done by Nikiforakis et al. [2] reveals that some websites today embed (and therefore trust) scripts coming from 295 different domains. The integration of Single Sign-On (SSO) services have been trending upwards and web statistics show that 55% of the 10,000 most popular sites use Google Analytics [6].

The widespread integration of third-party services raises a critical problem — how to ensure desired security, privacy and integrity properties for programs integrating third-party services. My research primarily targets two types of integrated applications:

1) Applications that rely on code and services provided by trusted third parties for critical security functionalities, e.g., authentication and authorization. In this case, the third party is trusted to be benign and its goal is to work together with its integrator to meet the security and privacy goals of the application. The goal for an outside attacker is to exploit logic flaws and violate the intended security properties of the integrated application. To address this threat, my observation is that vulnerable implementations often stems from misunderstandings between the integrator and service provider. To bridge the gap between the two parties, I developed techniques to explicate service SDKs, i.e. model concrete and abstract system components, run model checker and discover missing security assumptions and bugs in the SDKs. I also developed an automated vulnerability checker that leverages existing knowledge of common pitfalls and multiple heuristics to further help developers identify flaws in their implementations.

2) Applications that embed untrusted third-party components for non-security-related purposes. In web application contexts, such third-party code often provides add-on functionality such as analytics, advertising services, and social network widget features. The embedded scripts often run as the same principal as their host application and have full access to all client resources. These services may be malicious themselves or compromised by another party to distribute malicious content. To this end, I developed dynamic instrumentation approaches to help developers understand, monitor, and restrict third-party script behaviors, ensuring the application's security, privacy, and integrity properties.

## Protecting against outside threats

For applications embedding trusted third-party services, it is important that the developer of the integrated apps understands the assumptions upon which secure use of the service relies. Wang et al. [3, 4] showed that even trivial and subtle misunderstandings may bring disastrous vulnerabilities in the integrated application.

Therefore, finding these gaps in understanding is a key task to eliminate vulnerabilities. I developed an approach [5] to systematically find security-critical assumptions required by the service provider while working as a research intern with Dr. Shuo Chen at Microsoft Research. The first step of this approach is to model the relevant behavior of the system and ignore unimportant system details such as UI interaction and underlying network implementation. For black-box components of the system (e.g. identity provider server side), I analyzed their documentation and observed their behavior by crafting probe requests to create functional-equivalent models. Adding to the challenge, certain system parts may not have a concrete implementation at all. To handle this, abstract modules are created and I modeled these parts using non-deterministic function calls to imitate all possible actions they might take. After security assertions are inserted into the model, I ran a model checker that outputs paths that may violate the assertions. These violating paths lead to the discovery of potential bugs and missing assumptions in the real world. With the help of this process, the developer or service provider can fix the bug and add necessary assumptions. After these improvements are reflected back into the model, I ran the model checker again to continue this iterative approach. The process is done after all security critical components of the system have been modeled, and the model checker cannot raise a violating path. We call this entire approach to study an SDK *the explication process.*

I explicated three SDKs using this approach and found dozen authentication and authorization vulnerabilities. Several missing implicit assumptions are deemed serious enough to receive Facebook bug bounties and change the OAuth 2.0 specification. For example, we found a serious bug in Facebook's PHP SDK that would accidentally reveal the OAuth application secret (which must only be known to Facebook and the application owner) when a malicious client sends a sequence of carefully-crafted requests. Adversaries in possession of this secret may obtain user information and permissions on the victim application's behalf, or possibly mount impersonation attacks on the victim application.

To better understand the real-world implications of the discovered missing assumptions, I built a scanner named SSOScan [8], to automatically check applications for vulnerable SSO implementations. SSOScan can be used by an application marketplace such as Facebook's App Center or Google's Play Store. Contrary to general fuzzing or testing techniques, my proposed approach takes advantage of previously known vulnerability information to help automate and guide the tests in an efficient fashion. As an example, observing the presence of a specific type of OAuth token and replacing it with a fake token in the SSO process can accurately detect the app's vulnerable status. To dynamically observe the test application's behavior and mount simulated attacks against it, one of the main challenges SSOScan needs to address is to automatically walk through the user registration and SSO login process. SSOScan does this by using a series of clever heuristics: for example, our preliminary evaluation statistics show that majority of the login button on a website is visible, and is located in the vicinity of the upper right corner of the page. Using heuristics like this to weight candidates differently improves the automation success rate and testing time required for each site significantly.

I used SSOScan to study the twenty thousand top-ranked websites for five SSO vulnerabilities. Of the 1660 sites in my study that employ Facebook SSO, over 20% were found to suffer from at least one serious vulnerability.

## Protecting against embedded untrusted services

For applications that integrates untrusted third-party services, I target a different goal — understanding, monitoring, and limiting third-party scripts' access to the host application and preventing exfiltration of sensitive data to untrusted hosts. My approach to this scenario is two-fold: a blacklist-based approach [7] that limits what the third-party scripts must not do, and a whitelist-based approach that help developers understand and monitor their behavior [9].

To limit third-party scripts' accesses, I implemented a modified browser that understands and enforces fine-grained access control policies at the level of DOM nodes [7]. It isolates JavaScript execution contexts and restricts accesses to part of the hosting page from third-party scripts while also allowing these scripts to access the rest of the information to maintain functionality. The required security policies may be provided by site administrators. However, to offer a complete end-to-end solution, I believe that it is necessary to auto-generate policies and reduce the burdens on site administrators. I developed an automatic policy generator which infers sensitive information by comparing response differences across sessions with different credentials. While this approach often successfully reveals most private information, it could also result in many false positives including advertisements and social widgets that the third-party scripts need to access. After spending reasonable efforts to reduce false positives, I determined that the black-list policy may not be the best design to work with the policy generator.

Alternatively, I developed another modified browser named FireInspector [9], that compares the third-party script accesses to a resource whitelist and outputs the violating ones. Similar to the previous instrumented browser, FireInspector also intercepts API calls to access critical resources, such as DOM and cookies. FireInspector can also conveniently visualize the violations, which further helps the site administrators understand all third-party scripts behavior when they use an empty resource whitelist. They may then decide whether it is safe to embed the script or choose an alternative service provider.

As opposed to automatically identifying private information, the whitelist policy generation for FireInspector requires marking the third-party contents on the webpage, which gives us opportunities to improve upon the false negative-plagued blacklist policy generation. To this end, I built a policy generator that proposes policy candidates to site administrators based on access violations provided by FireInspector. The generated policies can also be visualized and presented for easier inspection. My evaluation for the policy generator reveals that it can effectively generate whitelist policies that capture most third-party behaviors for embedded scripts originating from 25 different third-party domains in 72 out of 100 test websites.

# Future Work

Over the next few years, I will continue to focus research on two general directions on which my current and previous works have focused — improving the security, privacy and integrity of integrated applications, and ease deployment for such security enhancement mechanisms. I firmly believe that these two objectives must go together to create realistic, end-to-end solutions.

**Extending the explication process.** As for improving security properties of applications embedding trusted third-party services, so far I have examined only one particular type — single sign-on service primarily on web applications. With the advent of various wearable devices and internet of things I believe the variety of such services will increase rapidly. I plan to extend the explication process to other platforms such as mobile devices and other services such as payment APIs and file sharing services. How to define the security properties, adjust and apply the explication techniques to new services and platforms remains to be the key challenges. I also plan to extend the vulnerability scanner to work on applications of multiple platforms.

**Self-checking SDKs.** In addition, my current works in this direction are mostly solutions from the view of an outside security service, and in the future I plan to attack these problems from service provider and end user's perspective. To address the root cause of this issue from the service provider's side, I will investigate the possibility of creating self-checking SDKs for trusted services. Such SDKs should be able to check its integrator's implementation for common pitfalls and may inform the developers or even do self-repairing.

**Cooperative testing.** From the user side, tools like easy-to-use browser extensions used for cooperative vulnerability testing to complement a tool like SSOScan — recording real user interactions and replaying them to simulate attacks can be effective against sites which SSOScan cannot automatically test. I believe the combination of these solutions together will make a much more secure mashup application environment.

**Using auto-generated authenticated sessions.** As a side product, the automated user interaction tools I built and plan to build will enable me to explore new opportunities for large-scale evaluations on many security mechanisms. Take SSOScan as an example: Many previous works have only evaluated their mechanisms using unauthenticated sessions due to the huge human effort required to register accounts and log in at test websites. Such is also true for many commercial web vulnerability scanners. Their results obtained this way miss coverage on authenticated pages, which is especially unconvincing because the goal is to address security and privacy concerns in the first place. After all, only minimum damage can be done in an anonymous session. SSOScan fills in this void by offering the ability to generate many authenticated sessions in fast and automated fashion, and I am eager to know how the increasing in testing coverage may affect the evaluation results of past and future works.

**Improving policy generation.** As for protecting applications against potentially malicious third-party services, I believe that the advancement and emphasis of security mechanisms have a significant lead over automatic policy generation in today's research. My future plan is to narrow this gap and focus primarily on improving automatically generated policies while requiring less human intervention. Looking at this problem from the perspective of the service provider, I will investigate how to incorporate program analysis techniques such as taint tracking on server code [1] to infer private or public information as it ships content to the client side. To further improve policy coverage, multiple integrators/users may collaborate and generate policies together. However, privacy challenges may arise due to the information exchange and must be addressed.

I believe the end product for a security mechanism is to get deployed in the real world. Particularly, one of the biggest hurdles against security mechanism deployment is when we ask the "beneficiaries" to do too much. It is essential that changes proposed to existing infrastructures be limited to minimum, and that the necessary changes be automatically generated as much as possible. I will make sure my future works to continue to address this while I develop better techniques to secure third-party service enabled applications.

# References


[1] Jonathan Burket, Jenny Cha, Austin DeVinney, Casey Mihaloew, Yuchen Zhou, and David Evans. Unifying Data Policies Across the Client and Server. In *Poster session of the 20th USENIX Security Symposium (USENIX Security)*, 2011.

[2] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. You Are What You Include: Large-Scale Evaluation of Remote JavaScript Inclusions. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012.

[3] Rui Wang, Shuo Chen, and XiaoFeng Wang. Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012.

[4] Rui Wang, Shuo Chen, XiaoFeng Wang, and Shaz Qadeer. How to Shop for Free Online – Security Analysis of Cashier-as-a-Service Based Web Stores. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy*, 2011.

[5] Rui Wang, Yuchen Zhou, Shuo Chen, Shaz Qadeer, David Evans, and Yuri Gurevich. Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization. In *Proceedings of the 22nd USENIX Security Symposium*, 2013.

[6] Wikipedia. Google Analytics Popularity. `http://en.wikipedia.org/wiki/Google_Analytics#Popularity`.

[7] Yuchen Zhou and David Evans. Protecting Private Web Content From Embedded Scripts. In *16th European Symposium On Research In Computer Security*, 2011.

[8] Yuchen Zhou and David Evans. SSOScan: Automated Testing of Web Applications for Single Sign-On Vulnerabilities. In *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[9] Yuchen Zhou and David Evans. Understanding and Monitoring Embedded Web Scripts. In *submission to the 35th IEEE Symposium on Security and Privacy*, 2015.