Defacing Facebook: A Web 2.0 Case Study
Adrienne Felt, University of Virginia

Rich technologies such as AJAX and Adobe Flash have popularized the use of the Web as a dynamic and interactive interface. Browsers are increasingly acting as operating systems, with code from multiple sources sharing the space of a single page on the client side. Gadget aggregators, such as Google’s Personalized Homepage and Windows Live, allow users to place dynamic third-party content in the context of the aggregator’s Web page. The aggregator provides the application with data, and the third-party application provides the user with a relevant, useful feature that the host aggregator lacks. (An example is a weather forecast localized to the user that he or she places on a homepage or profile.) In this situation, the aggregator cannot trust the third-party code but must still communicate with it and attempt to police its actions. Companies mitigate the potential for legal liability by stating that they are not responsible for the actions of third-party applications but simultaneously depend on the trust of satisfied customers for revenue. This work examines the implications of this new web security model and presents a case study on the gadget platform adopted by the wildly popular social networking site Facebook (facebook.com). The aim is to use this example to provide insight on the process of creating a secure gadget aggregation environment.

Facebook’s business model embodies the profitable trend of collecting user data and using it to create targeted ad revenue. Users enter their favorite books and music to find new friends with similar interests, and Facebook sends them ads relating to their preferences. Web sites that rely on ad revenue must draw and hold these ad-clicking users, and third-party applications deliver varied and fashionable entertainment without the cost of development. In May 2007, Facebook launched the Facebook Platform to enable the creation of applications that are closely integrated into the main Facebook Web site. Access to Facebook’s social network is exceptionally useful: users post names, education and employment information, contact information, marital status, friend relationship details, and more. According to Facebook, they had over twenty-five million members as of February 2007 and are the number one site for photos in the United States. Due to the amplification effect of the social graph, applications can reach a large number of users in a short period of time. For example, the application iLike gained three million users the first week it was available.

Applications are given access to user data through Facebook Markup Language (FBML) and Facebook Query Language (FQL). Users can add restricted applications to their profiles; these gadgets must be written in FBML and “pushed” through an API call that checks for adherence to FBML standards. As a security measure, JavaScript is not allowed on profiles. (Instead, dynamic user interaction is handled by “mock AJAX” API calls.) Alternately, users can access full-fledged applications on “canvas” pages where the arbitrary third-party code is isolated in an iFrame. Facebook attempts to preempt potential problems by manually screening applications before adding them to the official application directory; however, this practice is unscaleable, and applications may be altered after the inspection. Additionally, they crawl and automatically monitor applications to attempt to check that they comply with the Developer Terms of Service (which notably state that user information should not be retained past the expiration of a session key).

This research endeavors to identify new attacks enabled by the Facebook Platform, with the goal of understanding the threats posed by open web applications and developing design guidelines to prevent them. I am examining both policy decisions (e.g., allowing applications to store user data) and implementation details (i.e., are the policies properly enforced). The policy evaluation seeks to determine if the API can be abused in an unexpected way that allows attacks against the user or access to protected data. Tied to this, I will address the data mining and spam privacy concerns. The implementation testing includes an inspection of the API for resistance to cross-site scripting and an assessment of the effectiveness of the crawling/monitoring process. The proposed poster will present the results of this analysis and discuss their implications. Web service platforms are new, developing, and exciting, and this work considers important security points applicable to current and future mashups.