APPLICATION COMMERCE: SECURITY CHALLENGES AND WORKFLOWS

Vijay Anand, Jafar Saniie, and Erdal Oruklu
Department of Electrical and Computer Engineering
Illinois Institute of Technology,
Chicago, IL 60616, USA

Abstract— Application commerce refers to the economics related to the creation and distribution of software applications to the users of a computing platform. The need for a security infrastructure by the Application Programming Interface (API) makers for application commerce is necessitated by i) user privacy and trust concerns for the application, ii) the warranty concerns of the application maker, iii) the concerns from regulatory bodies (Recording Industry, Wireless Regulatory bodies) on content consumption and iv) revenue generation potential for the application maker. In this paper, we outline and compare the security infrastructures of existing API makers. The challenges posed by the security requirements and current solutions are discussed. We also present a workflow for application developers within the constraints set by the application framework provider for ease of development and distribution.

***Index Terms*—'Chain of Trust', 'Security Policy', 'Application Commerce', 'Design Flow'**

I. INTRODUCTION

Application creation and distribution mechanisms have evolved from the traditional models with the evolution of networks and mobile technologies [1], affecting the sale and consumption of applications. In this paper, we examine existing application development frameworks and their security mechanisms on various parts of the application execution stack. We identify the workflow for such an application development and distribution infrastructure, and show the security challenges associated with it. Finally, we propose changes to the workflow for seamless integration.

An application development framework or API (Application Programming Interface) refers to an execution framework within the constraints of the underlying stack. The application framework provider who is generally referred to as the API maker can be classified as either *Native, Value Addition* or *Web/Content* API maker, depending on their positioning software execution stack.

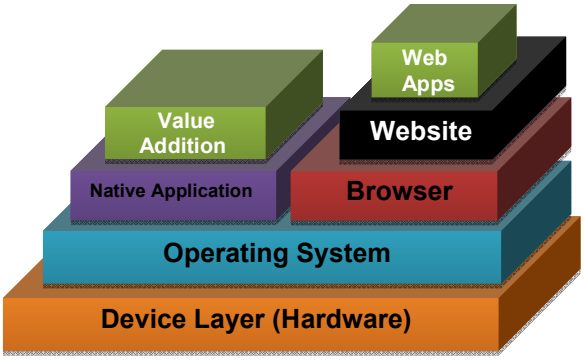

Figure 1. Application execution stack

Figure 1 shows the different types of frameworks that exist today:

Case I: Native Application

This class represents application framework (API) wherein the framework is created on the operating system part of the application execution stack. This is the closest an API ever comes to the hardware [2], hence, the term native applications. It mimics the traditional application framework. This part of the framework can also control secure services on a device and hence is powerful with respect to security. An API maker can thereby control which secure services can be made available to applications. An example of applications created in this model is Apps for iPhone which are hosted at Apple store [3].

Case II: Value Addition Applications

This class represents application development framework (API) wherein the API is created over the native applications part of application execution stack. Applications created from such an API are centered on value addition of services for the native application. Examples of such applications are add-ons [4] for Firefox or Chrome. Although Firefox is a browser, it is still a native application in context of the application execution stack.

Case III: Web/Content Applications

This class of application represents application development framework (API) wherein the API is created over the content of the application execution stack highlighted by Web Apps [5] in Figure 1.

Therefore, these applications are not hosted on a local computer but when content is accessed via any native or value added application these applications are activated and can be consumed. The API's in this model are created around the access of ownership of content. An example of such a model is that of App Directory [6] by Facebook.

This paradigm shift in application development opened new frontiers as highlighted by a Gartner [7] report but also introduced new security challenges [8] and requirements around application creation and distribution methods which are highlighted below:

- An application development framework or API within the constraints of the security [9] and quality of the underlying stack in the execution stack
- A secure application execution mechanism [8][9]
- A secure channel of distribution
- A monitoring and revocation mechanism[10]

In the next section, we show several real-world examples from existing application commerce frameworks.

II. Application Development Framework And Workflows

A typical workflow for application development [11][12] from the third party application developer is shown Figure 2. The various phases of application development are:

A. *Design Phase*

In this phase the application developer referred to in the Figure 2 as third party developer [12] constructs a design based on their knowledge base to create an application. The design phase is where the application creator exercises their intellectual property to generate revenue.

B. *Develop*

In this phase the third party developer creates the application based on the developer agreements [12]. This is where the developer may have to register with the API maker to leverage their intellectual property within the confines of the developer agreement.

C. *Test*

This phase marks the quality control phase [11][12] for an application maker. Quality control generally would refer to the phase where the application maker tests the performance, stability, user experience etc… issues within an application.

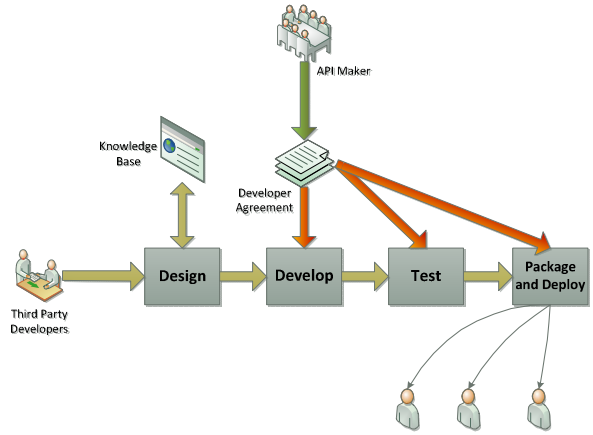

Figure 2. App distribution from the perspective of third party app maker

D. *Package and Deploy*

In this phase the application maker packages an application within the API maker's requirements and deploys for consumption [11] by the users of the application. This is the final phase for an application maker.

The three different classes of API makers based on the application development framework are compared below in Table I:

Table I. Comparison of developer workflow aspects for App Development

	Native Application: Apple iTunes	**Web Application: Facebook Apps**	**Value Addition Applications: Firefox Add-ons**
Developer Agreement	Cost based	Free	Free since it is Open Source
Test Distribution	Limit of 100	Yes	Yes
Hosting Server	Apple App Store	App - Directory	Add-ons
Monitoring	Yes	Yes	User Controlled

In the next section we see how the security framework of this application development affects the workflow.

III. Security Framework For Application Development

The need for security in an application framework is based on protecting assets in the computing service; privacy of users; having control over application distribution revenues from sales of application directly or indirectly; and reputation of the application [9]. To create a secure framework the API makers needs to create a control mechanism. This framework is based on the cryptography and the concept of chain of trust [13].

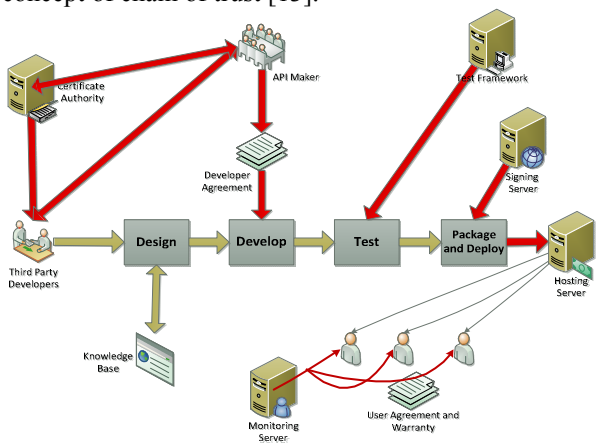

Figure 3. App Distribution Control Mechanism by API Maker

Chain of trust in this paper is defined as "*the transfer of immutable trust from one computing process to another such that the immutability of the trust factor is preserved during and after the transfer of trust*". This can be a static chain of trust or a dynamic chain of trust [14][15]. A static chain refers to a root of trust embedded in the device which can be chained out to induce trust. A dynamic chain refers to a root of trust which can vary and the chain of trust of can is induced from this. The chain of such a trust mechanism is guaranteed by cryptographic strength [16]. Cryptography provides the extension to the immutability factor down the hierarchy of the chain. Figure 3 shows the app distribution control mechanism which is described below.

A. Agreements and Security

One of the most overlooked issues in a security framework is the developer and user agreement. To create a developer and user agreement, the API maker needs to understand the assets in the system and the various regulations under which the API operates. The threat model of the system is the most critical element before drawing any developer and user agreements. Warranty agreements are drawn based on threats that cannot be reasonably countered within a price point. To create such an agreement, the first step is to identify the assets for the API's. The various assets are:

- The asset for the end user is privacy of user data.
- The asset for the device maker is the root of the trust chain such that the device maker can control the operation of the device with various service updates.
- The asset for an application maker is the revenue generated by selling the application in a way that the application cannot be tampered with, so as to bypass the revenue mechanism.

Based on this asset identification, threats are modeled for the system. The threat modeling techniques can be either threat trees [17] or the Petrinets [18]. Once threats which are beyond control of the API maker but directly affect the API maker are identified various agreements are framed.

B. Certificate Authority

The role of the certification authority [19] is to create application maker authentication and authorization. A certification authority is an entity in this workflow which houses the root key. It provides certificates with keys for an application developer that registered with the API maker to create application software. Other responsibilities of certification authority are to sign the application software once it is submitted to the authority and to revoke certificates if vulnerabilities or exploits are found in the application. The certification authority has to therefore support a revocation list of certificates which is used to determine whether to install and execute the respective applications.

Therefore, a third party app maker has to register with the API maker before accessing the API's to create applications. The registration process generally involves the usage of asymmetric encryption via an X509 certification mechanism. The reason to use asymmetric encryption is to allow the third party application maker to distribute an application without distributing the private keys. The control which the API maker puts in here is to prevent super-distribution of the application.

C. Testing Framework

The next aspect of a security framework which the API maker creates is that of a test framework [20]. The test framework generally carries the guarantees that the API maker would expect for a quality of an application. These guarantees are rooted in memory profiling, static code analyzers and integration tests with the API. This framework thereby helps to quantify some quality assurance metrics for the API

maker before the third party application can be released.

D. Signing Server

The next aspect of the framework is the signing server [19] which the API maker has to create for validation of an application such that it conforms to the developer agreement. This is necessary step for the third party app maker to get validation from the API maker to get acceptance. The other bearing of such a framework is that of control of super-distribution scheme by the API maker.

E. Hosting

The next aspect for control is revenue generation which the API maker gets from the distribution and hosting server generally portrayed as app store [11]. This allows the API maker to sell third party apps, and provides a framework for revenue sharing platform between the application maker and the API maker.

F. Monitoring and Revocation

The final aspect of control is that of the monitoring and revocation mechanism wherein the API maker monitors usage of the applications and revokes [10] offending applications.

A comparison of the major application vendors is shown in Table II.

TABLE II. COMPARISON OF SECURITY FRAMEWORK FOR APP DEVELOPMENT

	Facebook[6]	**Apple[3]**	**Firefox[4]**
Agreements	User, Developer	User, Developer	User, Developer
Certificate Authority	None, HMAC-SHA2 signatures	Managed by Apple	No explicit CA, But signing supported [21]
Testing Framework	No automated framework	Developer Oriented framework	None
Signing Server	None	Apple Server	None
Hosting	Any apps	Approved apps	Any apps with trust level indicator
Monitoring and revocation	Primarily User Driven	Automated	User Driven

As can be seen in the above comparison, every API framework has some desired characteristics over other. Apple has one of the most stringent requirements for application distribution. On the other hand, features like trust level indicators in Firefox add-ons are useful for app users.

In the next section, we propose a unified workflow for application development to address the key issues of application development and security concerns.

IV. PROPOSED DEVELOPER WORKFLOW FOR APPLICATION DEVELOPMENT

For an efficient workflow, both the ease of using a developer workflow and security aspects of development have to be addressed. In this section we propose a new workflow as shown in Figure 4, representing every aspect of the application development process. The chief aspects of this workflow are:

- Creation of application development framework (API) based on the application objectives and regulatory bodies requirements.
- Creation of application development agreement and user agreement.
- Creation of application testing framework
- Developer Registration by the API maker resulting in a unique identity referred to as the publisher ID. This maybe revenue driven.
- Download of the application development framework (API) to create applications.
- Create and Test the application within an application testing framework based on the API
- The test applications should be signed and they can be either locally signed or signed by a signing server. A signed application during test phase mimics an application in real world.
- Send signed application for approval. This phase gives a cursory check on the stability and quality of an application.
- Host application for consumption by users
- Provide interface to share revenue for app usage
- Monitor the usage of an application
- Revoke application if necessary

V. CONCLUSION AND FUTURE WORK

In this paper, we studied existing workflows for various application development framework providers for third party application makers. We compared these workflows and proposed an application development workflow. As computing systems move to the cloud, we aim to focus on the creation of secure workflows for development in the cloud for future work.

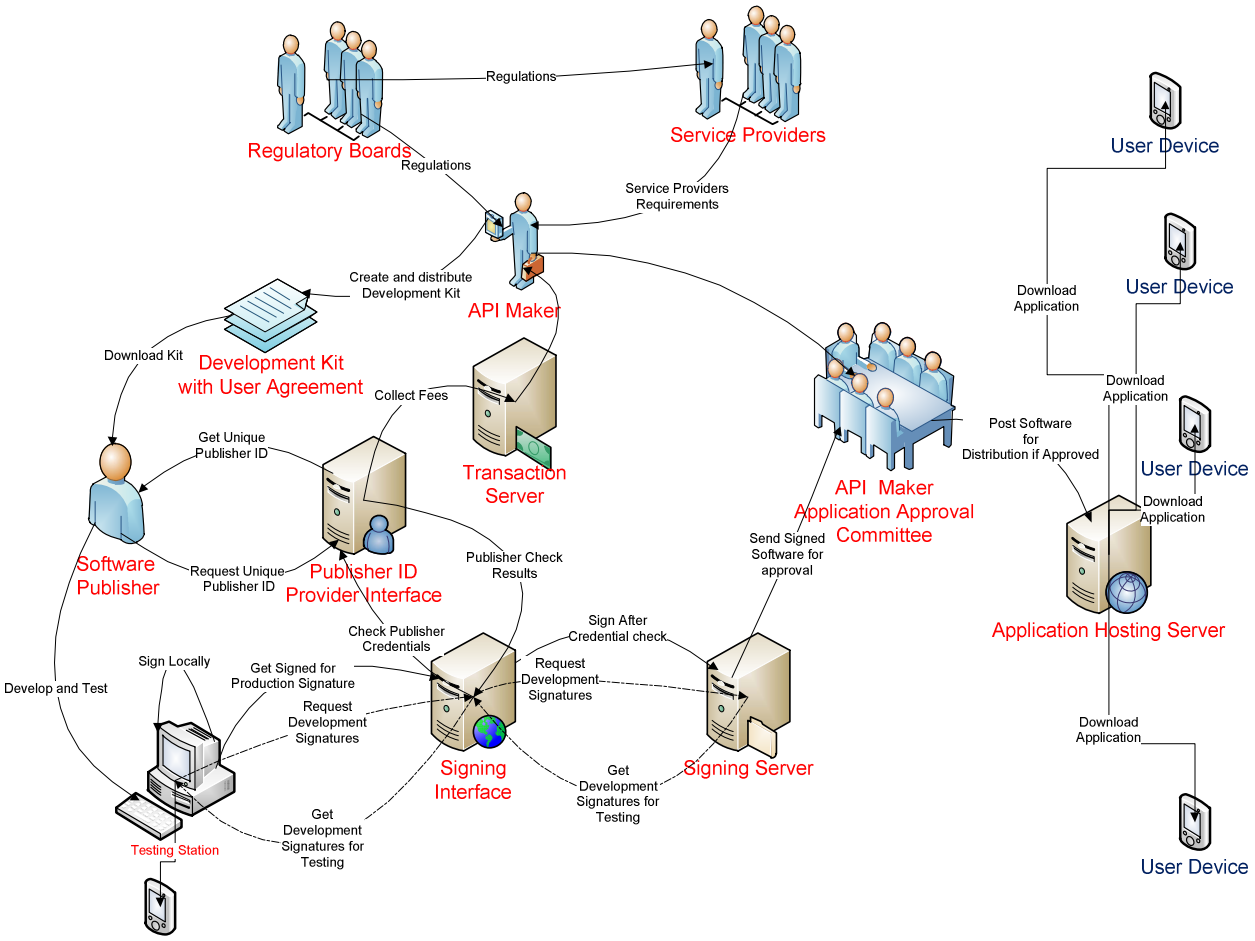

Figure 4. Secure developer workflow for application distribution

REFERENCES

[1] J. J. P. C. Rodrigues, M. Oliveira, B. Vaidya, "New trends on ubiquitous mobile multimedia applications", *EURASIP Journal on Wireless Communications and Networking*, vol. 2010, January 2010.

[2] P. Shenoy, S. Hasan, P. Kulkarni, K. Ramamritham, "Middleware versus Native OS Support: Architectural Considerations for Supporting Multimedia Application", *Real-Time and Embedded Technology and Applications Symposium*, 2002.

[3] Apple App-Store: http://www.apple.com/iphone/apps-for-iphone

[4] https://addons.mozilla.org/en-US/firefox/

[5] A. Ginige, S. Murugesan, Guest editors' introduction: "Web engineering—an introduction." *IEEE MultiMedia*., pp. 14-18, 2001.

[6] http://www.facebook.com/apps/directory.php

[7] Gartner: 185B mobile app downloads by 2014, http://www.betanews.com/joewilcox/article/Gartner-185B-mobile-app-downloads-by-2014/1296068736

[8] J.D. Meier, "Web Application Security Engineering", *IEEE Security & Privacy*, pp. 16 - 24, July/August 2006.

[9] E. Gudes, M. S. Olivier, and R. P. van de Riet, "Modelling, Specifying and ImplementingWorkflow Security in Cyberspace", *Journal of Computer Security*, vol. 7, no. 4, pp 287-315, 1999.

[10] V. Atluri, and W.K.Huang, "An Authorization Model for Workflows," *Computer Security – ESORICS, Springer*, pp. 44–64, 1996.

[11] L. Aouad, T. Kechadi, S. Petiton, "Persistent Locality Management of Scientific Application Workflows" 13th *IEEE International Conference on Computational Science and Engineering*, pp.404-409, 2010.

[12] L. Baresi, F. Casati, S. Castano , M.G. Fugini, I. Mirbel, and B. Pernici, "WIDE Workflow Development Methodology", *WACC ACM*, pp. 19-28, 2, 1999.

[13] W. A. Arbaugh, D. J. Fubert, J. M. Smith, "A Secure and Reliable Bootstrap Architecture" *Proceedings. IEEE Symposium on Security and Privacy,* pp. 65-71, 1997.

[14] B. Kauer, "Authenticated Booting for L4", November 2004. Available from: http://os.inf.tu-dresden.de/papers_ps/kauer-beleg.pdf

[15] B. Kauer, "OSLO: Improving the security of Trusted Computing", *USENIX Security Symposium*, article no. 16, 2007.

[16] A. K. Lenstra, and Eric R. Verheul, “Selecting Cryptographic Key Sizes”, *Journal of Cryptology*, vol. 14, pp. 255-293, 1999.

[17] V. Saini, Q. Duan, V. Paruchuri, "Threat modeling using attack trees", *ACM:Journal of Computing Sciences in Colleges*, vol. 23, no. 4, pp. 124-131, April 2008.

[18] D. Xu, and K. E. Nygard, “Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets”, *IEEE Transactions on Software Engineering*, vol. 32, no. 4, pp. 265-278, April 2006.

[19] N. Kapidzic, and A. Davidson, “A Certificate Management System: Structure Functions and Protocols”, *Proceedings of the Symposium on Network and Distributed System Security*, pp.153-160, 1995.

[20] K. E. A. Negm, “Design, implementation and testing of mobile agent protection mechanism for MANETS”, *The 3rd ACS/IEEE International Conference on Computer Systems and Applications*, 2005.

[21] Firefox: Signing an API: https://developer.mozilla.org/en/Signing_a_XPI