Advanced Science and Technology Letters
Vol.58 (Security 2014), pp.56-62
http://dx.doi.org/10.14257/astl.2014.58.12

Security Verification of the Programs in webOS app Store

Vyacheslav Belenko[1],Dmitry Zegzhda[2], Petr Zegzhda[2], Semen Kort[2], Vladimir Chernenko[1], Woosung Kim[1]

[1]LG Electronics inc., Korea
[2]Saint – Petersburg State Polytechnical University, Russia

Abstract. The article discusses the approach to application security analysis in the mobile store developed for LG Electronics webOS. There are investigated the features of static security analysis applied to the program written using JavaScript. We propose the description of the model intended to the malicious software identification as a pattern recognition system. This model specifies the various methods of software static security analysis.

Keywords: mobile application, Javascript, webOS, vulnerability, virus, static security software analysis, taint analysis, signature analysis, machine learning

1 Introduction

WebOS operating system based on Linux and designed for mobile phones and consumer electronics such as smart TV. LG Electronics is one of the webOS owners. The main feature of webOS is the capability of programs written in HTML and JavaScript to access system functions. Thus, the typical application consists of the HTML markup and the pieces of Javascipt code both written by the developer.

The analysis of the software put into the app store should identify the following security problems:

1. Software containing harmful code. The application may contain some hazardous code such as a virus or a worm. Such applications can be implemented using the mentioned programming languages. For example, the JS.Pros like fan worm [12] was developed in JavaScript, and is distributed via shared disk resources and file sharing applications.
2. The diverse application vulnerabilities. Vulnerability can be described with the following characteristics: the error caused the vulnerability, the impact of the vulnerability exploitation the authorization required to exploit the vulnerability (the attacker's rights determining the scope of possible actions), the attacker's location (describes the physical access needed for an attack of the system), etc. To investigate the actual types of vulnerabilities we should consider the model of the application executed by the browser or downloaded from the network server.

When the application is downloaded from the store it's program code is loaded from the local data store but not from the server. This changes the structure of the application shown in the Figure 1 with the client module only remaining. This signifi-

ISSN: 2287-1233 ASTL

cantly reduces the typical security problems caused by the vulnerable server application as it will be discussed further.

2 Related Work

The security investigation of the programs developed in the programming language JavaScript is quite relevant for the Web applications security. The typical security problems of the Web applications are discussed in the materials of OWASP foundation [1-3]. These studies provide the list of the security problems to be considered while testing software, as well as a set of requirements for secure coding. However, the relevant materials are mainly focused on the security of the server software as well as on identifying the typical vulnerabilities rather than on malwaredetection. This is not consistent with the objective of this work. The possibility of using machine learning techniques for WEB applications security analysis is discussed in [4]. The set of attributes to describe malware was defined by an expert, and algorithm J48was used for inductive learning. In [5] the set of attributes used to describe malicious and normal software was formed using the machine learningtechniques, thenBayesian classifier was applied for learning and recognition. The combination of data distribution analysis in the server module of the application and dynamic analysis in the client module is discussed in [6]. In [7], there is a static analysis of JavaScript code compliance to the language subset $\text{JavaScript}_{\text{SAFE}}$, with the programming language Datalog used for the analysis. The [8] provides the method of the static analysis of data distribution with the partial use for the dynamic program security analysis. Paper [9] gives the analysis of data distribution by the control graph for the applications containing client and server modules.

3 The static approach to software security analysis

3.1 The requirements for the applications

The applications under the static software security analysis are developed by LG programmers and invited application developers. Because of this all source code is available for analysis. To make the program security analysis more effective the requirements to the programs in store should be defined as follows:

1. all applications must indicate the links to the resources downloaded to the user's device;
2. if an application has links to external resources, it should comply with the following security policies to ensure the adequacy of the analysis:

– obtain fragments with the link to perform the static analysis;
– ensure that the link leads to the manufacturer website;
– get guarantees from the manufacturer on the compliance of the application code accessible by the link to the application code provided for the analysis;

3. in case if the manufacturer modifies the application code the application should be re-checked using the appropriate analysis.

Due to the varied kinds of problems with software described above (malware and programs containing vulnerabilities are known to have the different characteristics) several approaches to static software security analysis should be used.

3.2 Static software security analysis as a machine learning objective

In this section we consider the static software security analysis as a machine learning objective. The model of machine learning for malware detection can be formulated as a set

$$(D, \Sigma, F, K, S, R, P, C) \quad (1)$$

with the following terms:

- D is a program to be checked by the malware detection system (input). Typically D = (I, Data), where I = $\{i_0, i_1, ..., i_n\}$ is the partially-ordered sequence of instructions. The set Data = $\{d_0, d_1, ..., d_m\}$ is defined as a set of arrayed data that is not represented as instructions;
- Σis a finite set of program types indicating whether the program is normal or abnormal (output); in case of two binary response (normal or malicious program)∑={N, M};
- F is therepresentation of the program enabling to determine the class of software. Representation can fall into the program class that describes the sequence of system calls, CFG programs, a set of bytes, etc.
- S is a feature selection algorithm; it returns the feature set F

$$S(D, \Sigma) \to F \quad (2)$$

- R is an algorithm for representing the program in feature space

$$R(D) \to F \quad (3)$$

- P is an algorithm responsible for generating informationin the knowledgebase K;

$$P(D, \Sigma, \mathrm{F}) \to K \quad (4)$$

- C is a malware recognition algorithm

$$C(D, \mathrm{F}, \mathrm{K}) \to \Sigma\mathrm{C} \quad (5)$$

This model describes the process of malware recognition and can be instantiated depending on the types of the malicious software recognized, and the corresponding algorithms of recognition patterns. This paper identifies the two types of differently originated software security problems and proposes to use two detection algorithms. One of them should be used to detect malicious software, and the other - to identify typical vulnerabilities. This clarification enables to identify software flaws in terms of security for two models.

3.3 Malware detection model based on machine learning

The machine learning recognizes the malware by identifying the set of signs. Our main assumption is that such set exists. The model mentioned in the previous section can be specified in the following way:

- D is a program in JavaScript and HTML;
- Σ_{RPS} is a malware or normal program (different malware classes are not considered in this work);
- $F_{RPS}$is a set of signs enabling to distinguish normal software developed in JavaScript and HTML from malware; this set of signs is formed by experts and includes the following: a large number of string operations, machine codes in strings, iframe tags, off-screen elements, elements with a small footprint, the relationship between terminal and non-terminal symbols, entropy of strings, etc.The signs considered in paper [4] and falling into the category of the signatures of typical vulnerabilities; they are not discussed in this paper because they should be detectedby other methods;
- K_{RPS} is a set of rules that describe malware with F_{RPS} signs;
- S is a feature selection algorithm which is not used since feature selection is done by the expert;
- R_{RPS} is an algorithm of the program representation in F_{RPS} space;
- P_{RPS} is an inductive learning algorithm C4.5 with the database of malware containing both well-known and synthetic samples; it builds a set of rules K_{RPS};
- C_{RPS} is a result of security evaluation according to the set of rules;

3.4 The model of vulnerable software recognition based on machine learning

Typical vulnerabilities occur in normal software due to programming errors. As a result, they do not differ significantly from normal software in feature space detecting malicious software. Therefore, we should use a different approach to identify typical vulnerabilities in applications.

First of all it is necessary to determine what types of vulnerabilities can be found in the software developed for WebOS. According to the papers [1 - 3, 11] we should distinguish the following types of vulnerabilities related to client-side applications:

1. Code insertion. Code is inserted whenever the attacker sends untrusted data to the interpreter as part of a command or query and forces the shell to execute malware commands.

2. Attacks on Same Origin Policy (SOP) vulnerabilities, including attacks on Cross-site Request Forgery (CSRF) and DOM XSS.
3. Attacks on vulnerabilities of the JavaScript sandbox.

These problems do not include all typical Web application vulnerabilities due to the lack of a server-side, which leaves the typical attacks such as stored and reflected XSS [1-3] beyond consideration. The signatures of typical vulnerabilities are described in the table below in terms of language constructs JavaScript and HTML.

Now we can specify the machine learning model for the pattern search of typical vulnerabilities as follows:

- D is a program written in JavaScript and HTML;
- Σ_S is software containing typical vulnerabilities or a normal program which type is determined by its vulnerability;
- $F_S$is a description of typical vulnerabilities in the form of regular expressions;
- $K_S$is a database of typical vulnerability signatures; the table gives the set of the signatures of language phrases which may cause vulnerabilities; the set can be complemented with the list of signature expressions associated with the attacks on the browser [11]. The list of such expressions depends on browser and must be replenished as new vulnerabilities of the browser arise;
- $P_S$is an algorithm for describing the attacks signatures with regular expressions;
- $R_S$is an algorithm for the program representation; it is not used because the analysis is based on source texts;
- $C_S$is an algorithm for determining the regular expressions in the source program; the signature detected in one of the structures listed in the table causes the alarm, which can launch the search for code deficiencies;
- S is an algorithm of feature selection which is not used.

Signature search in the program code can detect vulnerable code fragments. Unfortunately, this method has a large number of false positives because it detects such code flaws that cannot be exploited. In order to specify the obtained results, it is necessary to analyze data distribution by the program graph described hereinafter.

The analysis of data distribution by the program graph (Taintanalysis) is a kind of data flow analysis exploring the distribution of values obtained from untrusted sources (methods and parameters) to the scope of sensitive to security operations. We assume in this work that any user input is untrusted and operations listed in the table are sensitive in terms of security. Then the analysis results in the assessment of possible impact of the user-defined input on the retrieval of typical vulnerabilities.

The analysis of data distribution is parameterized by the set of rules characterized by:

- the source specification (the line of code which inputs untrusted data into the program);
- the receiver specification (the line of code which executes the operation critical in the terms of security);
- the test specification (the line of code that verifies and corrects untrusted data).

Vulnerability is referred to as an untrusted information flow from the source to the receiver without verification. Source points are determined in the input program. Apart from traditional WEB applications this paper does not consider the server side. As a result all the URLs do not have a variable part depending on the server. Input

into the program written in JavaScript and HTML5 is carried out in the following ways:

- the address fields: using the parameter of the document.URL, or using the parameter window.location.href;
- forms of the user-defined input : the prompt method, document fields with the attribute INPUT (with the emphasis on hidden parameters);
- local storage : cookies, Local Storage API Objects;
- cross-server communication: Server-Sent (SSE) event processing, event processing.

To describe this problem in the terms of machine learning problem we represent the program graph as a context-free grammar. Therefore, we can specify the machine learning model for data distribution analysis by the program graph as follows:

- D is a program written in JavaScript and HTML;
- $\Sigma_S$is software that contains typical vulnerabilities or normal program; in case of the vulnerability its type is determined;
- $F_S$is a description of the typical vulnerabilities as regular expressions; the definition of the user-defined program input can be performed using context-free grammar;
- $K_S$is a database of typical vulnerability signatures; the definition of the rules allows to eliminate vulnerabilities and to normalize data;
- $P_S$is a definition of normalization rules as the restrictions on the input alphabet of the user input language;
- $R_S$is as licing algorithm which uses the context-free grammar to describe the relationship between the source and the receiver specifications;
- $C_S$is an algorithm of context-free grammar recognition for the programslice;
- S is a feature selection algorithm, which is not used.

4 Conclusions

The article describes the approach to the static software security analysis of the applications in webOS app store. The article discusses the following problems:

- the requirements for the developers of applications for the app store;
- the types of unsolicited software in terms of security and its codes;
- the common approach to the static software security analysis of the programs for webOS app store;
- the model describing static software security analysis as the pattern recognition;
- the methods to identify unsolicited software in terms of security based on the proposed model.

To check the security of the program and put it on the webOS app store we should ensure the following requirements :

1. the developer fulfilled the requirements for the developed program;
2. the analysis of the program for the detection of malicious software is fully complete;
3. the program is verified for the typical vulnerabilities.

The experimental studies have shown first and second type errors at the level of 0.5%. The suggested method is planned to be updated by the use of dynamic software security analysis.

References

1. Website materialswww.owasp.org
2. OWASP Secure Coding Practices Quick Reference Guide», The OWASP Foundation, 2010
3. M.Zalewski: The tangled WEB. A Guide to Securing Modern Web Applications», No Starch Press, 2012
4. D. Canali, M. Cova, G. Vigna : Prophiler: A Fast Filter for the Large-Scale Detection of Malicious Web Pages», ACM Press, 2011
5. C. Curtsinger, B. Livshits, B. Zorn, C.Seifert «ZOZZLE: fast and precise in-browser JavaScript malware detection», SEC'11 Proceedings of the 20th USENIX conference on Security, 2011
6. S. Guarnieri, B. Livshits. GULFSTREAM : Staged Static Analysis for Streaming JavaScript Applications, USENIX, 2010
7. S. Guarnieri, B. Livshits: GATEKEEPER: Mostly Static Enforcement of Security and Reliability Policies for JavaScript Code», SSYM'09 Proceedings of the 18th conference on USENIX security symposium, 2009
8. O. Tripp, O. Weisman «Hybrid Analysis for JavaScript Security Assessment», ESEC/FSE'11: ACM Conference on the Foundations of Software Engineering, 2011
9. S. Guarnieri, M. Pistoia, O. Tripp, J. Dolby, S. Teilhet, R. Berg «Saving the World Wide Web from Vulnerable JavaScript», ISSTA'11: ACM International Symposium on Software Testing and Analysis, 2011
10. Website materials http://www.symantec.com/
11. Website materials http://www.cve.org/

