

Assuring Software Security Against Buffer Overflow Attacks in Embedded Software Development Life Cycle

Chul Su Park, Jae Hee Lee, Seong Chae Seo, Byung Ki Kim
Department of Computer Science, Chonnam National University,
300 Yongbong-dong Buk-gu, Gwangju, Korea 500-757
{av3616, jaehee, scseo, bgkim}@chonnam.ac.kr

Abstract— The Current attack techniques against embedded systems are used more sophisticated. Security has been the subject of intensive research in the area of embedded system. Embedded system developers should consider the security technology which protects, mitigates, and removes the vulnerabilities throughout system development life cycle. This paper is intended to introduce embedded software analyzer to challenges involved in designing secure embedded systems against buffer overflow attacks. We attempt to provide a approach specifying of embedded software security requirements in the analysis phase of SDLC. It provides to understand, analyze, and specify the security requirements at an early phase of the software development.

Keywords— embedded software, SDLC, security requirement, modelling, buffer overfolw

I. INTRODUCTION

Current attack techniques against software are used more sophisticated. A number of software security incidents are increasing exponentially. In 2008, buffer errors account for 10.0(563/5,632) of the vulnerability reports from CERT advisories. Buffer overflow attacks cause security problems to embedded systems and general system[13].

Security is a requirement for an increasing number of embedded systems ranging from low-end systems such as PDAs, wireless handsets, networked sensors, and smart cards, to high-end systems such as routers, gateways, firewalls, storage servers and web servers[11]. Also, networked embedded systems will be subject to increasing security concerns. Embedded systems often provide critical functions that could be attacked by malicious entities.

An ever increasing range of attack techniques for breaking security such as software, physical and side-channel attacks require that the embedded system be secure even when it can be logically or physically accessed by attackers. We will concentrate on software attacks.

Software in embedded system is a major source for security vulnerabilities. Three factors of security risks in software are complexity, extensibility, connectivity[11].

Software in embedded system is a major source for security vulnerabilities. Three factors of security risks in software are complexity, extensibility, connectivity[11].

These conspire to make managing security risks in software a major challenge. Vulnerabilities have been found in almost every type of Software. Attackers can hack into systems by exploiting software defects such as buffer overflow, design flaws, etc.

Embedded software security's best practices leverage good software engineering and involve thinking about security early in the software development life cycle.

The idea of making secure software is to protect software from attack. Secure embedded software is still rarely taken into account throughout the entire software life cycle; security is often an afterthought, bolted on late in development, with little thought to what threats and exposures exist[7]. The Software Engineering Institute(SEI) study indicates that most software contains numerous defects during the development process[1].

To improve software security, software developers use models in the early phases of software development. McGraw[5] describes one set of best practices how software practitioners can apply them to the various software artifacts produced during software development. Many researchers have recognized that a integrated security with the software development life cycle can improve overall software security. Some researchers understood that was always better to design for security form scratch. Misuse case[4] and abuse case[6] are approach to addressing security threat analysis. These approaches concentrate on interactions between the application and its attackers who seek to violate its security. Security use cases[2] should be used to specify requirements that the application shall successfully protect itself from and its relevant security threats. But, for building security requirements, software developer cooperates with security team. Software developer and manager need the security technology which protects, mitigates, and removes the vulnerabilities.

This paper describes to assure software security against buffer overflow in embedded software development life cycle. To build secure software in SDLC, we describe a approach to specify security requirements at an early phase of the software development. We understand that software at highest level of

abstraction will tend to have the same cause of vulnerabilities. We are to tackle the threat at its roots.

II. EMBEDDED SYSTEM SECURITY

A. Buffer overflow in embedded software

Buffer overflow attacks have been causing serious security problems for decades. With more embedded systems networked, it becomes an important research problem to defend embedded systems against buffer overflow attacks. Buffer overflow attacks are classified into stack smashing attacks and function pointer attacks.

Buffer overflow vulnerabilities appear where an application reads external information into a buffer using vulnerable function. The possible places that have overflow vulnerabilities include stack, heap, or any other places that store variables.

B. Vulnerability Modelling

Software security aims to reduce exposure by reducing the number of vulnerabilities in software[7]. But software developers do not use security failure data, particularly, to improve the security of software that they develop. So Software developers should be aware of vulnerabilities.

Information about known vulnerabilities can already be found in several repositories or databases, such as ICAT, Bugtraq, etc. A taxonomy of software security errors can help software developers and security practitioners understand the common coding mistakes that affect security. It is to help developers avoid making common mistakes and more readily identify problems whenever possible.

The software vulnerabilities have originated from misunderstanding and the inconsistency of security requirements. Information systems being built and managed today are prone to the same or similar vulnerabilities. We need to classify security vulnerabilities. Some researchers[5] have been used in security failure data to protect and mitigate security vulnerabilities. It helps to find root of the security error and method of security error prevention, etc. Bishop describes that similar vulnerabilities are classified similarly and every vulnerability has a unique, sound characteristic (set of minimal size).

C. Approach to solve buffer overflow attack in SDLC

Traditional methods of developing software security are focused on how to write secure code[9] and how to design security mechanisms[7].

An approach to solve buffer overflow is to ask programmer to always do boundary checks or to use the safe programming language at implementation phase in SDLC. The static checking method uses the strategy to detect the vulnerabilities by analysing the code by software tools at testing phase in SDLC. The dynamic checking method detects buffer overflow vulnerabilities during the program execution. The longer vulnerability lies dormant, the more

expensive it can be to fix, and adding insult to injury, the programming community has a long history of repeating the same security-related mistakes.

Software specification, design, code defects are injected by software developers and common development practice must be changed to processes that produce software with very few defects. As Figure 1 shows, McGraw[11] specifies on set of best practices and shows how software practitioners can apply them to the various software artifacts that produced during software development. He shows the key security activities in relation to standard developments activities and deals with security activities consistency. We can know that integrating security throughout the life cycle can improve embedded system security.

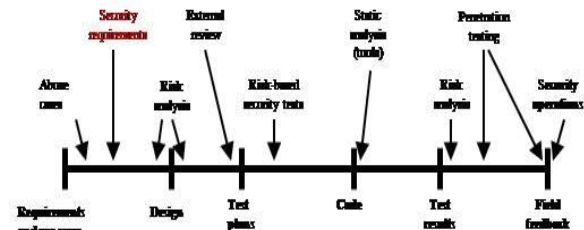


Figure 1. Software security best practices applied to various software artifacts.

The first activity for secure software against buffer overflow is specifications that define secure behavior exhibiting the security properties required. The specifications must define functionality and be free from vulnerabilities that can be exploited by intruders. So we need to specify security requirements.

D. Specifying Security Requirements

Security must be explicitly defined at the requirement phase of SDLC. The first necessity for secure software is specification of security requirements that define secure behavior exhibiting the security properties required. The ways to cover security requirements at the early phase in SDLC are Misuse case[4], Using abuse case[6], Security use case[2], Threat modelling, Security Patterns, etc. The analysis and specification of security is inherently difficult[1]. Security requirements deal with assets that must be protected and with the risk of these. These requirements should be appropriate and cost effective.

Misuse case and abuse case are an approach to addressing security threat analysis. These approaches concentrate on interactions between the application and its attackers who seek to violate its security. Also, it is an approach that is an extension to the syntax and semantic of the Unified Modelling Language. Misuse case is effective ways of analysing security threats but is inappropriate for the analysis and specification of security requirements[2]. Also, abuse case is used to analyze and specify security threats. These approaches help to secure web application, but it need to cooperate with security team, can't reuse artifacts.

III. ASSURING SOFTWARE SECURITY AGAINST BUFFER OVERFLOW ATTACKS

In order to improve the embedded system security against buffer overflow attacks, we need to specify security requirements at an early phase of the software development life cycle(SDLC). The analysis and specification of Security requirements have received considerably less attention.

The traditional techniques of the embedded system security do not adequately describe the security requirements. Also, software developers need to help security specialist and to know security expert knowledge.

It has to eliminate potential security flaws such as buffer overflow early in the embedded software development lifecycle. Every software at the highest level of abstraction will tend to have the same basic kinds of valuable and potentially vulnerable assets[5]. Our strategy to assure software security against buffer overflow is follows:

- 1) We identify the software threat location which can be existed buffer overflow vulnerabilities in embedded software. Seo, et al[12] describes a reusable diagram, which is constructed to be used to protect, mitigate, and remove vulnerabilities, which is located in software threat location.
- 2) We apply reusable diagram to the products in the analysis phase of SDLC. We describes process to apply reusable diagram to the products in the in the analysis phase of SDLC.
- 3) We apply to the rule for building the security requirements.
- 4) Finally, we specify security requirements.

We describe how you apply reusable diagram to embedded system, and how you specify security requirements. We model embedded system with UML(Unified Modelling Language). We focus on the specification of embedded security requirements. To identify threat location in software by attackers, we use that Seo, et al[12] describes a reusable diagram, which is constructed to be used to protect, mitigate, and remove vulnerabilities, which is located in software threat location. He classified threat location s into user input process, network process, data control process. Buffer overflow corresponds to user input process at threat location. So, we concentrate on process that applies reusable diagram to the products in the analysis phase of SDLC.

A. Modelling Embedded System Requirements

To capture functional requirements of embedded system, you model a use case diagram. Each use case provide to scenarios. A scenario is an instance of a use case, and represents a single path through the use case. You make sequence diagram model in relation to scenarios. A sequence diagram identifies the behavior within embedded system.

To find threat location, we use a sequence diagram. Threat location in sequence diagram is user input process, network

process, data control process. User input process is between actor and boundary class, network process is between control class in client and control class in server, data control process is between control class and entity class.

B. Applying reusable diagram to sequence diagram

In order to apply reusable diagram at threat location of buffer overflow, you need to find the assets that should be protected. The assets of embedded system consist of data and service. If a data should be protected, you mark the type of threat location at the sequence diagram. The threat location at the sequence diagram is as follows. The threat location of user input process is between boundary class and control class. The threat location of network process is between control class and control class.

Buffer overflow attacks are classified into the threat location of user input process. You can mark an applying point between actor and boundary class on the sequence diagram. Then, as illustrated in Figure 2, you apply reusable diagram to applying point on the sequence diagram.

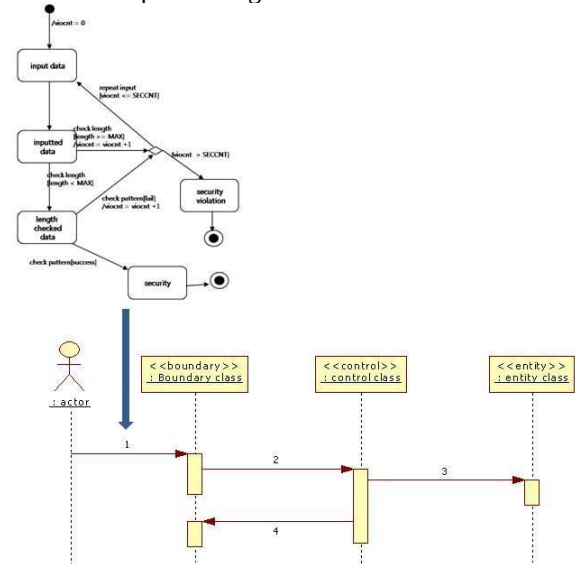


Figure 2. Apply reusable diagram to sequence diagram.

C. Specifying the Embedded System Security Requirements

To specify the embedded security requirements, you replace a data which is represented at reusable diagram with a real data which is used to develop embedded system. Also, you replace a guard condition which is represented at reusable diagram with a real guard condition which is used to develop embedded system. The rule of specifying the web security requirement is that a modified guard and event of reusable diagram must be specified the embedded system security requirements. On the assumption that real data is job data, the embedded system security requirements specified through proposed approach are as follows.

1. The length of input job data must check.
 - a. if the check of job data length fail, you add 1 to the security violation count.

- b. you must check the security violation count
- 2. The pattern of input job data must check.
 - a. If the check of pattern of job data fails, you add 1 to the security violation count.
 - b. you must check the security violation count.

IV. CONCLUSION AND FUTURE WORK

Embedded system developers have recognized that integrating security throughout the embedded system development life cycle can improve overall web application. This paper describes approach to find a threat location of buffer overflow in software, to specify embedded system security requirements through reusable diagram at threat location. It provides to understand, analyse, and specify the security requirements at an early phase of the embedded system development. It provides embedded system developer with usability, reusability about specifying security requirements. Also, it provides cost-effective security and improves the specification of web security requirements. But it can't resolve all embedded system security problems. For improving security, we need to use the static checking tools.

In the future, we would like to study the consistency between embedded system security requirement specification and static analysis.

ACKNOWLEDGMENT

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (KRF-2008-313-D01014)

REFERENCES

- [1] D. Firesmith, "Common Concepts Underlying Safety, Security, and Survivability Engineering", CMU/SEI-2003-TN-033, 2003.
- [2] D. Firesmith, "Security Use Case," Journal of Object Technology (JOT), Vol. 2, No. 3, May/Jun 2003, pp.53-64.
- [3] D. Firesmith, "Specifying reusable security requirements", Journal of Object Technology 3, 2004.
- [4] I. Alexander, "Misuse cases: Use cases with hostile intent", IEEE Software 20, 2003, pp. 58-66.
- [5] I. V. Krsul, "Computer vulnerability analysis", PhD thesis, Purdue University, 1998.
- [6] J. McDermott, C. Fox, "Using abuse case models for security requirements analysis", Proc. Annual Computer Security Applications Conference (ACSAC'99), 1999.
- [7] J. Viega, G. McGraw, Building secure software, Addison-Wesley, 2004.
- [8] M. A. Howard, "A process for performing security code reviews", IEEE Security & privacy, vol. 4, no. 4, July-Aug. 2006, pp. 74-79.
- [9] M. Howard, and D. C. LeBlanc, Writing secure code, Second edition, Microsoft, 2003.
- [10] P. Koopman, "Embedded System Security", IEEE Computer, 37(7), 2004, pp.95-97.
- [11] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, and S. Ravi, "Security as a new dimension in embedded system design", In Proceedings of the Design Automation Conference, 2004, pp. 753-760.
- [12] S.C. Seo, J.H. You, Y.D. Kim, S.J. Lee, B. K. Kim, "Building Security Requirements Using State Transition Diagram at Security Threat Location", CIS2005, 2005, 451-456.
- [13] Z. Shao, C. Xue, Q. Zhuge, M. Qiu, B. Xiao, and E.H.M. Sha, "Security Protection and checking for embedded system integration against buffer overflow attacks hardware/software", IEEE Transactions on Computers 55(4), 2006.
- [14] Z. Shao, Q. Zhuge, Y. He, and E.H.M. Sha, "Defending embedded systems against Buffer Overflow Via Hardware/Software," in Proceedings of the 19th Annual Computer Security Applications Conference(ACSAC 2003), 2003, pp. 352-363.