Resolving Security Issues of Cognitive Radio Networks

Sangook Moon

Mokwon University, Department of Electronic Engineering

E-mail : smoon@mokwon.ac.kr

ABSTRACT

The cognitive radio (CR) network has been studied in the form of open source by vast number of communities, and the potential expectation is very high since the CR is based on reprogrammable platform. However, as the peer-to-peer software has been abused, so high is the chance that the CR network can be abused public wide. Consequently, the benefit from the study of next-generation wireless network can be at risk because law breakers could abuse the CR. In this contribution, we analyze the issues and the problems of the CR and discuss an efficient measure against security attacks.

Keywords

Cognitive radio, security, next-generation wireless network.

Ⅰ. Introduction

With the advent of ubiquitous era, the number of wireless communications devices has been growing exponentially, thus poverty of resource of physical frequency arose as a serious problem. Accordingly, to avoid the traditional way of allotting wireless channels in fixed regions, cognitive radio has been introduced in which finding empty channels is possible by itself and communicate within its channels respectively [1].

In order to satisfy the needs of the users of increasing wireless communication, it is essential to secure enough frequency bandwidth. However, wireless frequency bandwidth for communication is finite, and most of it is occupied by primary users, which makes it difficult to assure additional frequency band. In fact, research by FCC and other institutes indicates that the efficiency of usage of preoccupied frequencies turned out as low as 30%, especially in suburbs and regions with low population density [2][3].

CR technology is created based on the software defined radio to utilize the ability of spectrum sensing to recognize available frequencies and choose to communicate, with special ability to update environmental parameters by itself continuously. To realize the CR in effect, studies on efficient channel sensing and channel hopping have been in progress. In this paper, I discuss a few issues on securities and suggest ways to resolve the problems in software aspect.

Ⅱ. CR Verification and Security Measurement in Software Manner

The verification method of embedded software in CR is different from that of traditional software in which testing is done throughout the whole system. Instead of inspecting every possibility of program code, the point is whether the CR can be guaranteed to comply with the regulation policy of the network. Compared to the traditional method which relies on debugging, CR can be verified as long as every CR device does not violate the policy generated by regulation entity. This way we are able to get rid of the potential malicious code, which could not have been removed in the traditional way. For the sake of this way of verification, I suggest two concepts in view of software.

A. Abstracion of a Program Based on New

Swarm-Intelligence

First, I suggest an abstraction of a program based on new swarm intelligence. That is, applying the Ant Colony Optimization (ACO) [4] to the abstraction technique, to enable discriminating of the part of a program code, which is expected to be examined first. ACO is an algorithm of recognition which finds the optimal answer through the collaboration and adaptation form of virtual ants, which emulate real ants' habit. Apart from its original purpose of creating the test data, in this paper, I modify and expand the ACO to simplify the process of path level abstraction for software verification. The basic concept is as follows. Given the characteristics of design under test, virtual ants follow the program flow. During the movement, the state space which is transferred by violating the order can be found. The trace followed by the virtual ants can be used as the initial program under-approximation. Experiments with thousands of program codes revealed that considerable amount of the code could be removed.

B. Code Level Metric for Evaluating the Weakness of Variables in a Program

Again apart from the traditional security measurement metric which focuses on the system level, I suggest a preemptive, cost-effective method in order to discriminate weak codes as soon as possible in the program design level. Focusing on the code level rather than the system level could reduce effort for finding and fixing the error which threatens security. By this concept we can remove weak points of code before the software takes its place to put the security level high and make malicious attack difficult.

Weak program variables are the key target of security attacks. Let's look at the code in Fig. 1. Scalar variables range, count and array variables deststr, nums are all the potential target of attack.

There are two scenarios in general form of attack. In the first case, such variable as deststr itself is the target of attack. In the line number of la, because strcpy function does not automatically take care of the boundary of arrays, variable deststr becomes overflow if the input buffer length of inputstr exceeds the size of assigned target buffer of deststr. In the

```
proc foo(char *inputstr){
      int range, count;
      char deststr[50];
    int nums[20];
    …
    range = 40;
la:   strcpy(deststr, inputstr);
lb:   for (int i = 1; i < range; i++) {
        nums[i] = fun1(nums[i-1]);
    }
    count+= fun2();
}
```

Fig. 4. An Example Code with Variables Exposed to Attack

second scenario, variables can be indirectly affected by attacks targeted to other variables such as range or count. According to the general structure of local variable stack, values of two scalar variables can be indirectly overwritten by overflow of neighboring array variables of deststr. However, not in all cases does changing the values of variables raise the security problems of a program. Rather, comparing with variable count, range is more important in the respect of security, because without the verification of effectiveness, the value of inappropriate variable range in line lb causes overflow of array variable nums. In the other way, if we change the value of count with malicious intention, the effect of which is trivial compared with what range affects.

We can learn from the example above that we can practically use the numbering the priority of risk degree of security in variable level in finding weak points in the program code. Another thing that we can learn is that we should secure precise modeling of program structure against the effect of malicious security attack as well as extensive analysis. Therefore, traditional simulation based attacking model achieved from the analysis of partial program behavior is not efficient for code level security verification. So I, in this paper, in order to derive the security sensitiveness in variable level, consider the frequency which is enough for passing the security property test and the security weight with regard to that. Suppose there are N security property variable spi (i is integer where 1 <= i <= N). Each spi is supposedly related to security weight swi. Suppose verification candidate variable v@l

(variable v in line l) is given, and assume this falls into either safe subset (security property sps1, 1 <= i <= Ns) or vulnerable subset (security property spv1, 1 <= i <= Nv) after the first stage of the algorithm, where 1 <= Ns, Nv <= N, and Ns + Nv = N. Now, the security vulnerability of variable v in line l can be calculated as (1).

$$varSec[v@l] = (\sum_{v_1}^{v_N} sw_{vi} - \sum_{s_1}^{s_N} sw_{si})/N$$
$$v : variable, \quad l : lane$$
$$1 \le Ns, \quad Nv \le N, \quad Ns + Nv = N$$
$$safe\ subsset : sp_{s1}, sp_{s2}, ..., sp_{sNs}$$
$$vulnerable \subset\ : sp_{v1}, sp_{v2}, ..., sp_{vNv} \quad (1)$$

As we can see in (1), vulnerability of a variable is determined by combinational security damage that affects the whole system security when the variable is attacked. We can make (1) a formula to verify as an element of program model checking. The more violating property a variable has, the more threatening effect it will potentially have.

Future work will include implementing a verification engine. We can now use model checking as the formal verification platform to check the vulnerabilities of the security in the program code.

III. Conlcusion

CR technology is expected to be applied as any form in wireless communications sooner or later. The XG project under progress by U.S. Pentagon relates CR networks to military equipment. War can occur anywhere around the world, and the military equipment with fixed frequencies is useless. If CR is available to the military equipment, the military operations can be successful making use of the empty slots of frequencies by CR technology. Besides, in the case of public disaster prevention communication, many frequency slots are necessary when disaster occurred, but minimum of them are required in usual case. Therefore, with CR technology, public safety bands can be occupied by normal users in ordinary times, but when emergency, CR devices with normal users detects the disaster and release the frequencies to the primary users so that frequency utilization goes high. In this way, CR technology is essential to future wireless network and is apt to weak to malicious attacks because of its cognitive characteristics. In this paper, I suggested two methods of resolving the security attacks in view of software. I expect both methods can be cornerstones to protect the smart cognitive radio networks. Future work will include implementing a verification engine. We can now use the proposed ways of model checking as the formal verification platform to check the vulnerabilities of the security in the program code.

Reference

[1] S. Haykin, "Cognitive Radio: Brain-empowered wireless communications," IEEE J. Sel. Areas Commun., vol. 23, no. 2, pp. 201-220, Feb. 2005.

[2] C Kim, “Cognitive Radio technology trends,” Weekly technology trends of Korea, Vol. 21, No. 4, August, 2006

[3] http://www.ofcom.org.uk/research/technology/research/emer_tech/cograd/

[4] http://iridia.ulb.ac.be/mdorigo/ACO/ACO.html