The Research on Attack Threat Sensing based on "Software Security Sensor"

Yuan Zhao[1,2], Qiangbo Liu[1],Dingyi Fang[1,2], Huaijun Wang[1,2],Cong Zhang[1,2]
[1]School of Information Science and Technology, Northwest University, Xi'an, China
[2]NWU-Irdeto Network-information Security Joint lab (NISL)
Corresponding Author: zhaoyuan19890807@126.com

***Abstract*—Aim at the different attack threats the running software faced with during reversing, a method of sensing attack threats based on "software security sensor "is proposed in this paper. And the instance has demonstrated that the method is effective and feasible. Drawing lessons from the thought of physical sensors, the code snippet which is used to sense the attack threats is called "software security sensor", SwSensor for short. Firstly, the attack threats and their features are analyzed; then the design of the corresponding SwSensors and the delimiting of sensed areas is discussed in detail; finally, the layout model based on the multi-level gateway in physical sensor network is described.**

Keywords- Attack threat; Sensing; Software Security Sensor; Code Blocks Dependency

I. INTRODUCTION

The software runs in white box attack environment[1], leading to the key information exposed to attackers further. Meanwhile, the continuous development of attack tools and methods makes the white box attack environment more dynamic, complex and unpredictable[2].

Traditional software protection methods, such as, encryption, obfuscation belong to the static protection because the protected software cannot adjust their execution paths dynamically.2011, Collberg put forward the idea of dynamic protection, who thought that the factors including speed, agility, unpredictability, vigilant monitoring, defense in depth, and renewability of defenses are all necessary to ensure long-lasting defenses[3].So, to make the protected software updated the defense with the changes of environment and dynamically game in the terminal with the attacker is a new idea to explore dynamic protection further. The first problem is how to make the running software sensed the attack threats in white box attack environment.

Reversing attack will cause the changes of elements in white box attack environment, which is the fundamental basis to sense attack threats. Usually, the reverse analysis process can be divided into three stages: the disassembly, dynamic and static analysis and decompiling[4], as shown in figure 1.The first stage is to disassemble the binary code into assembly code. The second is to understand the functions and semantics to get control flow, data flow and algorithm etc. The last is to decompile assembly code into source code. However, the differences among high-level languages result that the attack effect based on decompiling is not ideal. Attackers attack the first two stages usually by means of some mature tools, such as OllyDbg, IDA, to set breakpoints, memory dump, etc. Among them, the dynamic and static analysis is the process of human thinking, and does not change the elements of environment. Therefore, the attack threats that the running software is faced with in white attack environment can be divided into three types: debugging attack, dumping attack and tampering attack from the perspective of reverse analysis.

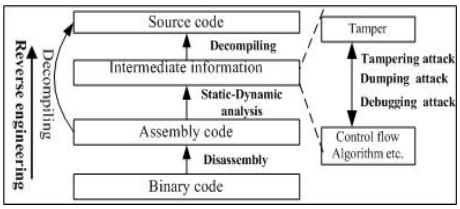

Figure 1. Reverse Engineering and Attack Threats.

II. RELATED WORK

There are some related research results at home and abroad.

- Anti-debugging techniques based on detecting the changes of environment elements.

Most of the current anti-debugging techniques have been used[5] in commercial protectors. Its principle is that some elements of the white box attack environment will be changed when debugging[6]. However, the current anti-debugging techniques are simple and can be removed by some plug-ins directly.

- Tamper-proof techniques based on multi-point guards.

Hoi [7] put forward using guards net for the integrity protection. Cappaert[8] improved this idea and proposed a new tamper-proof scheme based on cryptography. A part of code is encrypted, and the left parts are used to get the key. When executing, only the code to be running is decrypted and will be encrypted again after executing. The function call graph is used to establish the dependencies among parts. Until 2012, the idea that link all the guards to cycle chain to ensure the security of guard itself is proposed [9]. Although the ideas of guard net and code dependencies are very valuable, the technology itself still belongs to static defense.

- Sensing techniques with intelligent decision.

Arash Salehpour et al.[10] developed an intelligent guard which could monitor the changes of environment and make decisions on whether continuing to execute in the software life cycle. Artificial intelligence is introduced in this technique to make the terminal software be self-adjusting with the changes in the environment. Due to the intelligent

guard and protected software are separated, attacking on intelligent guard itself becomes a bottleneck.

III. ATTACK THREATS SENSING BASED ON SWSENSOR

A. Overview

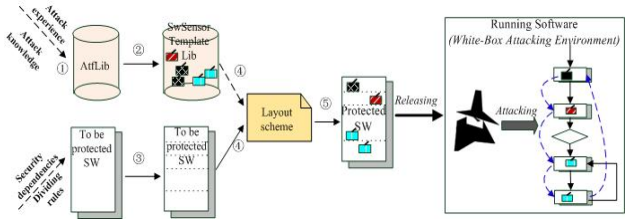

Figure 2. Mechanism of Sensing the Attack Threats.

We will introduce the basic concepts firstly.

B. Basic Concepts

- Sensed elements

Refer to the factors which will cause the changes of white box environment or affect the running software, such as, the debugger will modify the flag bit of Beingdebug. And the other factors, such as, the sequence of API, which can reflect the changes of software itself, also belong to sensed elements.

- Set of attack threats features

Attack threats can be divided into debugging, dumping and tampering. Different attack threats will change different sensed elements and the subset of which can identify different attack threats are called set of attack threats features.

- Security dependencies among code blocks

Refer to the software attributes used to identify the influence on other code blocks when a code block is being attacked, such as, invoking, data dependencies, etc.

- SwSensor

Assembly code used to sense the attack threat in running time, such as, the snippet code of integrity checking.

C. Method of Sensing Attack Threats

Step1. Collect the sensed elements of debugging, dumping and tampering based on attacking experience and knowledge. Marked as $f_{ij} = \{name, way, value\}$, f_{ij} represents a sensed element, *way* represents the usage of f_{ij}, *value* means the secure value of f_{ij}.

Construct the attack threat feature library as following. *DgThreat* refers to the debugging attack threat, *DpThreat* refers to the dumping and *TpThreat* refers to tampering attack threat.

$$AtfLib = \begin{Bmatrix} DgThreat: f_{11}, f_{12}, \ldots f_{1x} \\ DpThreat: f_{21}, f_{22}, \ldots f_{2y} \\ TpThreat: f_{31}, f_{32}, \ldots f_{3z} \end{Bmatrix}$$

Step2. Construct the corresponding software security sensors library, marked as *SwSensorLib*, according to *AtfLib*. The design principle is that judge whether the software is in a secure state or not by identifying the features of attack threats is normal or not. The details are shown here:

- Design the functional template of SwSensor, marked as $SwSensorTemplate = \{At, fset, Parameter_m\}$, *At* represents the type of attack threat, *fset* represents the subset of features, $Parameter_m$ presents the parameter list, such as, the start address of sensed area.
- "Feature value" is used to reflect different attack threats, *Feval* for short. The sensing algorithm is designed of randomly selecting several features and normalizing them to get *Feval*, marked as *Algorithm*.
- Use the deformation engine to diversify the different *SwSensorTemplate* to construct the SwSensorLib. $SwSensorLib = \{SwSensorTemple_i, Swsensorset_i\}$, $Swsensorset_i$ represents the set of Swsensors coming from $SwSensorTemple_i$.

Step3. According to the security dependencies and division rule among code blocks, delimit the sensed areas.

- According to the attack experience and knowledge, obtain the security attributes $CSAttr_i$ and form the security dependencies among code blocks. $CodeSecDep = \{CSAttr_1, CSAttr_2, \ldots CSAttr_n\}$.
- Delimit the sensed areas of the being protected software $SArea_i = \{CodeSecAttr_i, parameter_i\}$ then form $SwSensorArea = \{SwSArea_1, SwSArea_2, \ldots SwSArea_m\}$.

Step4. Generate the layout scheme based on multi-stage gateways model.

- Matching suitable SwSensor for the different sensed area and setting their parameters. Marked as $\forall SwSensor_w^k \in SwSensorLib, Swsensor_w^k \mapsto SArea_i$, $Swsensor_w^k$ is used to sense the state of $SArea_i$,
- Setting the parameters of SwSensors based on minimum coverage theory of graph. Regard the chosen SwSensors as the vertices of graph, the sensed scope as edge sets, construct associated matrix and adjust the sensed scope repeatedly to get the minimum coverage vertex set. And considering the security dependencies among code blocks, level classification is carried out on the chosen SwSensor and the multi-level gateway structure in accordance with self-security is constructed.

Step5.Reconstruct the protected software to make the SwSensors triggered in the appropriate time.

IV. INSTANCE AND EVALUATION

A. Attack Threats Lib

Extract several features of three attack threats below.

- DgThreat features: the execution time, the breakpoint;
- DpThreat features: the API sequence;
- TPThreat features: code checksum and control flow;

The description of attack threat library is shown in table 1.

TABLE I . DESCRIPTION OF ATTACK THREAT LIB

Attack Threat	Sensed elements	Name	Way	Sec_val
DgThreat	f11	ExeTimevar	When the process is being debugged, the debugger event handling code, step instructions etc will occupy CPU cycle. If it takes between adjacent orders far more than conventional, means the process is being debugged.	[t1,t2]
	f12	Breakpoints	Int3 breakpoint will modify a byte memory into 0xCC.	Not 0xCC
DpThreat	f21	APISeq	When fixing ImageSize, call for API sequence below: GetProcessBaseSize,GetProcessPath,CreateFile,ReadFile	Not this API sequence
TpThreat	f31	Checksum	Checksum is always same without tampering attack.	SUM
	f32	ControlFlow	Code the control flow and test if it is changed in runtime.	{a1a2…an}

B. *SwSensor Template Lib*

1) *Design of SwSensor Template*

Combine the different sensed elements of one attack threat to generate the multi-subsets of features.

TABLE II. DESCRIPTION OF SWSENSOR TEMPLATE LIB

Template	Attack Threat	Features	Parameter list
ST11	DgThreat	f11,f12	<startaddr,endaddr,val>
ST12	DgThreat	f11	<startaddr,endaddr,[t1,t2]>
ST13	DgThreat	f12	<startaddr,endaddr, 0xCC>
ST21	DpThreat	f21	<startaddr,endaddr,APIseq>
ST31	TpThreat	f31,f32	<startaddr,endaddr,val>
ST32	TpThreat	f31	<startaddr,endaddr,SUM>
ST33	TpThreat	f32	<startadr,endadr, {a1,a2…an>

For example, the two sensed elements of DgThreat from table 1 are combined to produce three subsets of features: {f11, f12}, {f11} and {f12}. Then, compute the "feval" of each subset. The second subset only includes {f11}, so its "feval" can be directly identified as [t1, t2], but the first subset includes {f11, f12 which do not belong to the same dimension and must be computed by normalizing.

2) *Sensing algorithm*

Sensing algorithm
Read fij from AtfLib
Normalize(fij) to get VAL
Compare VAL, parameters.val
If unequal Broadcast()
else Continue

The Normalize (fij) is defined by user. Broadcast () transfers the attack information.

We define the Normalize (fij) below:

①ST11:t2-t1+0xCC; ②ST12 :(t2-t1)* t1;

③ST13:0xcc; ④ST21:1111;

⑤ST31: SUM+a1+a2+…+an;

⑥ST32: SUM; ⑦ST33:a1+a2+…+an.

A part of pseudo codes of templates are shown in table 3.

TABLE III. PSEUDO CODE OF SWSENSOR TEMPLATES

ST11	ST12	ST32
mov eax,t2	Invoke GetTickCount	add ebp, checksum
mov ebx,t1	mov ebx,eax	mov eax, startaddr
sub eax,ebx	Invoke GetTickCount	for:
add eax,0xcc	sub eax,ebx	cmp eax, endaddr
sub eax,feval	mov ebx,eax	jg end
jz label	mul ebx	mov ebx, dword[eax]
continued	sub eax,feval	add ebp, ebx
label:	jns label:	add eax, 4
Broadcast	continued	jmp for
	label: Broadcast	end:

3) *Diversifying Swsensors*

Deformation engine[11] is used to tranform the SwSensor templates to different forms which are equivalent Semantics. As shown in figure 3.N-Iterative is the parameter to control the number of iterations of matching instruction pattern. It will generate mulitple SwSensor templates with different deformation level.

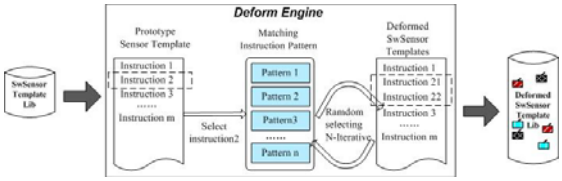

Figure 3. Deformation engine.

C. *Delimiting the Sense Area*

1) *Security dependencies*

We adopt the control dependency[12] as the security dependency.There are three atomic control dependencies based on the functions granularity[13],as figure 4 shows.

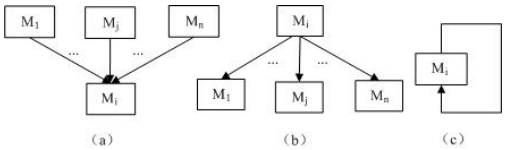

Figure 4. Three atomic control dependencies.

The weighted directed graph commonly is used to indicate the control dependencies among code blocks, as shown in figure 5. Among, $P \to Q$ represents P depends on Q, the weight means the degree of control dependency.

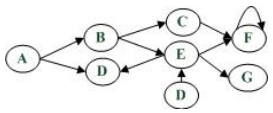

Figure 5. The control dependency among code blocks.

The degree of control dependency is computed here:

Step1: Daw the control dependency graph without weight;

Step2: List the paths starting from the node with zero in-degree and ending in that of zero out-degree;

Step3: The weight of one edge equals to the outdegree of head node plus the weight when it is tail node.That is, the weight of "Q→W" in "P→Q→W" equals to the outdegree of Q plus the weight when Q is tail node. For the ring node as the tail node,the weight of the edge plus 1.

Step4: Repeat step 3.

So, we analyze the weight of each edge in figure 5 here:

List all paths: ①ABCG,②ABDE,③ABDG,④ABDHH,⑤AE,⑥FDE, ⑦FDG,⑧FDHH.

And then, mark the weight from the node with zero indegree:①A2B4C5,②A2B4D8E,③A2B4D8G, ④A2B4D8H9H, ⑤A2E, ⑥F1D8E, ⑦F1D8G, ⑧F1D8H9H.

2) Assembly of Sensend Areas

Sort the code blocks according to control dependency with key code block from strong to weak, and then select first n related code block as the assembly of Sensend areas.

As shown in figure 5, it is assumed that "D" is the key block, sort the other code blocks according to the control dependencies with "D" from strong to weak:DEGHBF. In this case, select "DEGHBF" as the assembly of sesend areas.

D. Layout schema and Reconstrction

1) Matching SwSensors with code blocks

Choosing the appropriate SwSensor for different sensed area according to two principles:①Select SwSensor according to the code block's own features; ②Considering the performance cost,select the smallest SwSensor only if meeting the first priciple.

2) Layout model based on multi-layer gateway

It is to assign a value to the parameters. Like the left in figure 6.The control dependencies among the block codes aren't changed in fact and just increased several layers of SwSensor. The sensed attack information is passed to the higher layer, which is similar to physical multistage gateway.

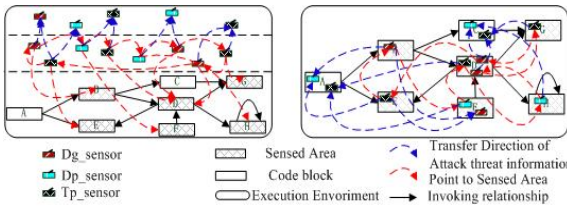

Figure 6. Layout model based on multi-layer gateway.

3) Reconstrction

The chosen SwSensors are embedded into the code blocks in the assembly instruction level to generate the protected software.Like the right in figure 6.

V. CONCLUSION

Method of attack threat sensing based on "software security sensor" is proposed in this paper. Explore what kind of attack threats are in the white box attack environment and their features in the process of reverse engineering. And then research on how to make the software sensed the different attack threats according to the features in runtime. However, how to store and pass the attack information safely is another key point in the research of self-adjusting, which is the next step of this paper.

ACKNOWLEDGMENT

This work is supported by the Natural Science Fundation of China(61070176, 61170218, 61272461), Research Fund for the Doctoral Program of Higher Education of China(20106101110018), Research and Industrial Project of Shanxi Province(2011K06-07).

REFERENCE

[1]De Mulder Y, Wyseur B, Preneel B. Cryptanalysis of a perturbated white-box AES implementation [M]. Progress in Cryptology-INDOCRYPT 2010. Springer. 2010: 292-310.

[2] Huang Yu, Yu Jian-Ping, et al. Monitoring Properties of Open Environments [J]. Journal of Software, 2011, 22(5).

[3]Collberg C. The Case for Dynamic Digital Asset Protection Techniques[J]. Department of Computer Science, University of Arizona, June 1,2011: 1-5.

[4]Zhao Yu-Jie, Tang Zhan-Yong, et al. Evaluation of Code Obfuscating Transformation [J]. Journal of Software, 2012, 23(3): 700-711.

[5]http://bbs.pediy.com/showthread.php?t=106143&highlight=%E5%8F%8D%E8%B0%83+%E8%B0%83%E8%AF%95+%E8%AF%95. 2010

[6]Gagnon M N, Taylor S, Ghosh A K. Software protection through anti-debugging[J]. Security & Privacy, IEEE, 2007, 5(3): 82-84.

[7]Chang H, Atallah M J. Protecting software code by guards [M]. Security and privacy in digital rights management. Springer. 2002: 160-175.

[8]Cappaert J, Preneel B, Anckaert B, et al. Towards tamper resistant code encryption: Practice and experience [M]. Information Security Practice and Experience. Springer. 2008: 86-100.

[9] WU Shao-jie, HE Rong-yu, et al. Study of Software Protection Method Based on Cyclical Guards [J]. JISUANJI YU XIANDAIHUA, 2012, 1: 161-165.

[10]Salehpour A, Etemad M, Nazarlu M M. Intelligent Guard: A Novel Approach toward Software Protection [M]. Informatics Engineering and Information Science. Springer. 2011: 449-460.

[11]Desai P. Towards an undetectable computer virus[D]. Citeseer, 2008.

[12]Cappaert J, Kisserli N, Schellekens D, et al. Self-encrypting code to protect against analysis and tampering[C]. 1st Benelux Workshop on Information and System Security (WISSec 2006), 2006:14.

[13] XU Bao-wen,Zhang Ting,et al. Dependence analysis of recursive subprograms and its applications [J].Journal of Computer, 2001, 24(11): 1278-1283.