Dynamic Taint Analysis for vulnerability Exploits Detection

Heping Tang Shuguang Huang Yongliang Li Lei Bao
Department of Network engineering
Electronic Engineering Institute HeFei Anhui China
{ tangheping2007, Shuguang,yongliang,baolei}@163.com

***Abstract*—Untrusted Data originating from network input and configuration files, causes many software security problems. Keeping track of the propagation of untrusted data in program runtime is the main idea of dynamic taint analysis for vulnerability exploits detection. In this method data from network user input and configuration files were labeled as taint. In virtue of data flow analysis we design taint propagating algorithm, and define several taint detection policies for security-critical function which used taint data in dangerous ways that could cause vulnerability exploit. A vulnerability exploit detection prototype system was implemented. In contrast to other taint analysis systems, our prototype system has higher accuracy and vulnerability exploits coverage and low workloads.**

Keywords—Dynamic taint analysis; Vulnerability exploits detection; Data flow analysis; Tainted scenes analysis

I. INTRODUCTION

A large number of security problems are related to input validation vulnerability, such as buffer overflows, format string vulnerability, forgery system calls, SQL injection, XSS and other vulnerabilities. Security analysis marks the entry of taint, and records taint propagation. The method which uses data flow method to discover and track the non-trusted data is known as the taint analysis.

Taint propagation analysis is originally inspired from the Perl taint model. In the Perl taint model, external data are tagged as taint including user's input, environment variables, files, etc. Taint data can't be used as the arguments of security related functions as local bash command, sending and receiving data functions.

II. DYNAMIC TAINT ANALYSIS

Dynamic taint analyses are widely used in computer security fields. It prevent a variety of attacks, including buffer overflow[1,2], string format vulnerabilities[3], SQL injection attacks [4,5], cross-site scripting attack [6], etc.

A few of general framework of taint analysis are available recently. Lam and Chiueh[2] introduce the general approach using virtual machine technology to implement taint marking and propagation but it need the target code to be recompiled, which isn't convenient for the third-party software and Libs.

Dynamic taint analysis is mainly used for detecting software vulnerability exploits. The common exploit method is overwriting attacks, which rewrite the return address, function pointer, the system call arguments and so on. Newsom, and Song[3] proposed methods based on dynamic taint analysis to prevent coverage attack. Others also focus on improving the efficiency of dynamic taint analysis, optimizing taint propagation. Suh [7] and Kong [1] respectively proposed a hardware mechanism for taint marking.

Dynamic analysis is also used to prevent SQL injection attacks. Attacker submits a malicious query statement to the database behind web applications. Most of the dynamic taint detection method could be used to detect SQL injection by tracking the propagation of SQLs in the program. Nguyen Tuong[6] presented a mechanism to analyze taint propagation on PHP based web applications.

III. DYNAMIC TAINT PROPAGATION VULNERABILITY DETECTION

Dynamic taint analyses for vulnerability detection focuses on taint propagation and exploit detection mechanism. The research of dynamic taint analysis is composed of three parts:

(1) Tags taint data and determine the taint data source;
(2) Taint data propagation process;
(3) Warning mechanisms.

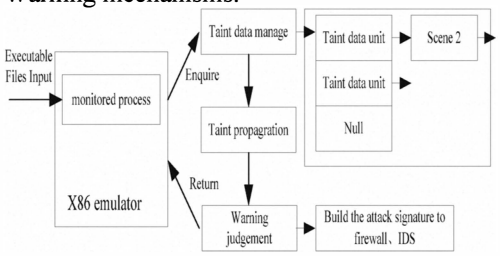

Figure 1. the framework of vulnerabilities detection based on taint propagation

The framework of vulnerability detection system based on dynamic taint analyses is illustrated in Figure 1.The entire detected system used as a monitor sub-procedure of X86 instruments. Once the running of program reaches a new block, X86 monitor interrupt the target program and converted binary instruction into RISC instruction, then deliver it to taint data management module. If an operand is tainted, the system allocates a new taint data unit to store detailed information.

If the security functions accept taint data, the warning module will sent an alarm and display the execution path from the entry to the unsafe function. For other functions operations, the monitor returns directly and waits until the next code block interrupts

IV. TAINT PROPAGATION ALGORITHM

Dynamic taint propagation algorithm

Tainted data are stored in linked list. So taint path from taint source to destination are able to be tracked by traversing through tainted data link. Unit in taint link records register name, content, taint type, assignment operation of taint data. Current taint collection is a snap of tainted data in process presently. Taint data are immediate cleared and eliminate from current taint collection when they are replaced by normal data.

Taintedlink: = (Node | Node: = <Tainedaddress, value, taintedtype, pparentTaint, operation>);

CurrentTaint: = (object | object: = <Tainedaddress, value, taintedtype>)

Taint analysis are carried out as follow steps: The assembly code blocks are handed in to taint propagation analysis. Each possibly tainted memory in processes are marked as taint during program execution. We check possibly tainted input such as argument, global variable and external input before a function call. According to dynamic taint propagation algorithms, the type of every assembly instruction are firstly identified (assignment/stack operation/arithmetic operation) and operands are extracted from instructions for further taint data calculation. Taint propagation algorithms are listed in figure2. Taking assignment operation as an example in taint calculating procedure, if source operand is tainted, the destination operand is tainted subsequently; otherwise the source operand is sanitized. The algorithm is illustrated in pseudo code in figure 3.

```
Algorithm: Taint_propagation
Input:AssemblyCodeBlocks
{
1  if((AssemblyCodeBlocks=ø)
2     exit ;
3  for each ci∈AssemblyCodeBlock{        //For each intruction
4     switch typeof(ci):
5        case: MOV|R*L|R*R|XCHG|STOS|LEA
6           e1 = LeftOperand(ci) ;
7           e2 = RightOperand(ci) ;
8           Taint(e2→e1)  ;              //taint propagation
9        case: PUSH*
10          e1 = [ESP] ;
11          e2 = RightOperand(ci) ;
12           Taint(e2→e1) ;
13       case: POP*
14          e1 = LeftOperand(ci) ;
15          e2 = [ESP] ;
16           Taint(e2→e1) ;
17       case: AD*,DIV*,MUL,NEG,SUB,OR,XOR,AND
18          e1 = LeftOperand(ci) ;
19          e2 = RightOperand(ci) ;
20           Taint(e2)→Taint(e1) ;
    }
}
```

Figure 2. Taint_propagation algorithm

```
Algorithm: Taint
Input:e2→e1
{
1  if( e1=ø ) OR ( e2=ø )
2     exit ;
3  if( e2∈TaintedObject )                  // e2 is tainted
4     e1.pparentTaint = e2 ;               // linked to its parent
5     Taintedlink.add(e1) ;                // add to taint link
6     TaintedObject = TaintedObject U e1 ; // add to current taint
7  else
8     if( e1∈TaintedObject )               //e2 is normal data
9        e1.taintedtype = ø ;              //e1 is cleared
10       TaintedObject = TaintedObject - e1 ;
}
```

Figure 3. Taint algorithm

A. *Taint propagation Analysis Algorithm*

The taint link and current taint collection are constructed in parallel during program execution by recording every possible tainted data unit. Taint link stores taint scenario information and details of taint propagation operation. Taint propagation analysis based on taint link and current taint collection includes: (1) taint reachability checking; (2) taint propagation boundary analysis; (3) taint clearance.

Taint reachability checking aimed at evaluating the possibility of using taint data in security-related function. Function accepts user-supplied data as parameters，such as an integer or a string, its vulnerable code can be triggered by user data. The algorithm of taint check rule is illustrated as follow. Taint data can be inferred by rules that whether the data is member of current taint collection.

```
Check function(arg1,…,argn)
   IF function is security-critical
      AND
      arg1∈CurrentTaint ∪ … ∪ argn∈CurrentTaint
   THEN unsafe
   ELSE safe
```

If the tainted data are able to reach security-related function after several transformations, warning module immediately prevents process from running by invoking an application exception. Taint tracking are used to discover the taint path. Taint path are deduced recursively from taint links which provides taint scenario information. Algorithm in figure 4 provides a method to find a path from taint argument to its origins.

```
Algorithm: TaitPathAnalysis
Input:TaintedArg
{
1  e=TaintedArg ;
2   path = ø ;
3   do{
4         path.add(e) ;          //add to taint path
5         e = e.pparentTaint
6   }
7  while(e.pparentTaint≠null)
```

Figure 4. Taint track algorithm

Taint propagation boundary analysis is mainly concerned on taint clearance. For example the length of user input is usually constrained before use. If the length of external string less than buffer, buffer over flow couldn't occur. Buffer over flow checking could be canceled. The early discovery of taint clearance makes dynamic vulnerability exploit detection more effectively.

Elimination of taint data is the ultimate goal of exploit detecting. Although we can discovery a few of software flaws manually, but automation is vital in large scale software production safety checking. Taint path analysis provides various taint clearance point and we could find a best position to place our clearance code.

B. Vulnerabilities exploit detection policies

Because of the existence of various software vulnerabilities, exploit detecting strategies are designed for each kinds of vulnerability types. Table 1 shows the detecting methods based on dynamic taint propagation.

TABLE I. VULNERABILITIES EXPLOIT DETECTION POLICIES

vulnerability	Taint Checking Strategy	comment
Directory traversal	File_function:=execute_file(path,)| Open(path,)|CanonicalizePathName(path,)Path:"\..\path" →"input"t	Tainted path parameter of file operation function, such contains "\..\".
String formatting	Printf(fmt,);fmt:"…(%n)t…"	Tainted format parameter of string output function, containing "%n"
SQL injection	SQL_query_function(query); Query:"…(sqlmetachar)…"	Tainted data in SQL query statement, containing '',execute'

V. EXPERIMENT

A. MS06-055 vulnerability analyse

MS06-055 is vector markup language vulnerability. An attacker could construct a specially web page or HTML e-mail to exploit the vulnerability. When a victim visits the web or check e-mail, malicious code could be executed on victim host. The vulnerability is located in the VGX.DLL. The procedure of copying the name of a method marked in html like '<v:fill method="XXXX"/>' where "XXXX" is the calling method name caused a buffer overflow vulnerability. In function sub_5AD02D1B, it allot 210h bytes buffer in the function stack for method name, but it did not compare the length of method name with buffer size. Attacker can construct a method name larger than the buffer size to trigger buffer overflow vulnerability. The path of taint propagation is illustrated in figure5. All the sub-function names are automatically create by IDA pro 5.0. Below the box, we record the location and key operation of taint data.

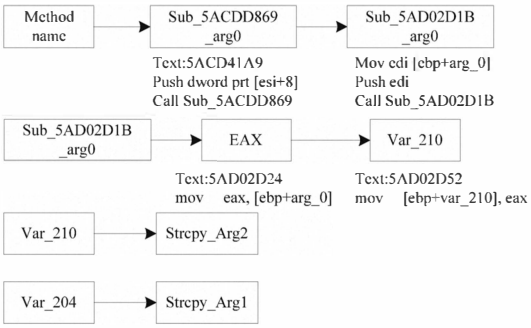

Figure 5. Taintedlink of ms06-055 vulnerability

If an attacker intends to exploit the MS06-055 vulnerability, he or she should inject taint data into the data transfers channel. The alarm module keeps monitoring the key memory location in taint path. All intent to exploit the vulnerability can be detected.

B. MS08-067 vulnerability analyse

MS08-067 is a directory traversal vulnerability in windows directory operation procedure, due to a service does not properly handle specially crafted RPC request. Once successfully exploiting this vulnerability, an attacker could remotely control victim system. Figure 6 is a taint propagation path of MS08-067.

RPC demos receives taint data form network request. Request data are copied to the directory variable 'Directory', then the length of file are calculated and stored in variable 'length'. An attacker can control the pollution data, and firstly create a file request as "\ x. \ .. \ yyyyyyyyyyyyy". The file name is transformed to standard Windows directory format inside function CanonicalizePathName (). Due to the existence of a logic error, the program constantly looks for "\" in stack. Meanwhile, the attacker constructs a residual image of "\" at the top of the stack, eventually leading to overwrite the return address. Vulnerability exploit detection system sent a warning, once it detects a string "\..\" in functions parameter. Vulnerability exploit detection based on the taint propagation analysis records the trace of taint data.

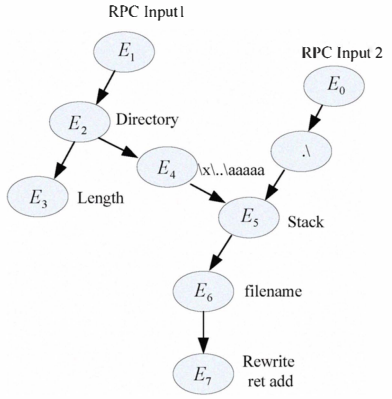

Figure 6. Tainted link of ms08-067 vulnerability

C. Experiment results analysis.

In order to verify the correctness of vulnerability exploit detection system based on taint propagation, we implement a

taint propagation analyses prototype. A comparison between existing vulnerability detection system and our prototype system is list in table 2, which shows that the method can effectively detect a variety of vulnerabilities, and provide detailed exploit scene information in virtue of program static analysis method.

TABLE II. THE COMPARISON DYNAMIC TAINT PROPAGATION SYSTEM FOR EXPLOIT DETECTION

Taint analyses	Description	Target	Vulnerability
TaintCheck[3]	Tag taint data, monitor target program, and tace taint data propagation.	binary file	Overwrite vulnerabilities exploit
Dytan[10]	Framework of taint analyses	binary file	Heap,Stack overflow,SQL injection
SQLCIVs	Syntax check on PHP source code	PHP web application	SQL Injection
Bounds checking[11]	Track data object in system, mainly focus on Bounds of buffers	C/C++sources	Buffer overflow exploit
Method in this paper	Analysis all the taint scene by data flow analysis, symbol execution	binary file	Heap,Stack overflow,SQL injection, format string,directory tarversal

VI. SUMMARY

Traditional access control policy has its limitation on discover the dangerous data. In SQL injection and cross-site scripting attacks, there is no difference between normal users and attacker except theirs request data. Other vulnerabilities exploit, such as buffer overflow, format string vulnerabilities are easy to evade policy-based detection mechanism. Attackers construct a specific attack sequence to trigger the vulnerability and ensure the attack data are accepted by vulnerable function.

In this paper, we propose vulnerability detection system based on dynamic taint propagation analysis and implement a prototype to verify our method. It does not require the source code and symbol information of target program. Target programs are running in virtue environment. Data from untrusty sources such as network, user input and configure files are tagged as taint data. Taint path and the collection of taint data are calculated by taint propagation algorithms.

All of data received from outside are checked by comparing with thousand of attack signatures. Vulnerability exploit detection focus on taint data which is probably used by security related function. Experiment result shows the mechanism improves the efficiency of vulnerability detection greatly.

REFERENCES

[1] Kong J, Zou C, Zhou H. Improving Software Security via Runtime Instruction-level Taint Checking[C]. In: ASID '06: Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability, 2006,18–24.

[2] Lam L, Chiueh T. A General Dynamic Information Flow Tracking Framework for Security Applications[C]. In: 22nd Annual Computer Security Applications Conference, 463–472.

[3] Newsome J , Song D. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software[C].In:Proceedings of the Network and Distributed System Security Symposium, 2005.

[4] Alfond W, Orso A, Manolios P. Using Positive Tainting and Syntax-aware Evaluation to Counter SQL Injection Attacks. In: SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering[C]. New York,USA, 2006,175–185.

[5] Pietraszek T, Berghe C.Defending Against Injection Attacks Through Context-Sensitive String Evaluation[C].IN: RAID 2005:Proceedings of Recent Advances in Intrusion Detection, 2005.

[6] Nguyen Tuong A, Guarnieri S, Greene D et al. Automatically Hardening Web Applications Using Precise Tainting[C].In: 20th IFIP International Information Security Conference, 2005.

[7] Suh G, Lee J, Zhang D, Devadas S et al. Secure Program Execution via Dynamic Information Flow Tracking[M]. In ASPLOS-XI:Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems, New York, USA, 85–96, 2004. ACM Press.

[8] Qin F, Wang C, Li Z et al. LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks. In: MICRO '06: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture[C], 135–148.

[9] Clause J, Li W, Orso A. Dytan: A Generic Dynamic Taint Analysis Framework[C]. In ISSTA'07:International symposium on software testing and analysis 2007 196-206.

[10] Chuang W, Narayanasamy S, Calder B. Bounds Checking with Taint-Based Analysis[C].2007.