A Dynamic Taint Tracking Optimized Fuzz Testing Method based on Multi-modal Sensor Data Fusion

Qianmu Li (qianmu@mail.njust.edu.cn)
Nanjing University of Science and Technology https://orcid.org/0000-0002-0998-1517
Shunmei Meng
Nanjing University of Science and Technology
Hanrui Zhang
Nanjing University of Science and Technology
Yaozong Liu
Wuyi University
Haiyuan Shen
Jiangsu Zhongtian Technology Co., Ltd
Huaqiu Long
Wuyi University

Research

Keywords: Fuzz test, Dynamic taint analysis, Dependence relationship, Grammar generation

Posted Date: February 14th, 2020

DOI: https://doi.org/10.21203/rs.2.23600/v1

Version of Record: A version of this preprint was published at EURASIP Journal on Wireless Communications and Networking on June 3rd, 2020. See the published version at https://doi.org/10.1186/s13638-020-01734-0.

Abstract

The safety of Industrial Internet Control Systems has been a hotspot in the information security. To meet needs of communication, a large variety of proprietary protocols have emerged in the field of industrial control. The protocol field is often trusted in the implementation of industrial control terminal code. If attackers modify the data of these fields using the protocol defect, the operation of the program can be controlled and the entire system will be affected. To cope with such security threats, academia and industry generally adopt fuzzy test methods. However, the current industrial control protocol fuzzy test methods generally have low code coverage, where unified description models are missing and test cases are not targeted. A method of fuzzification processing combined with dynamic multi-modal sensor communication data is proposed. To track the program execution, the dynamic pollution analysis is used to search for the input fields that affect the execution of the conditional branch, and capture the dependencies between the conditional branches to guide the grammar generation of test cases, which can increase the chances of executing deep code. The experimental results show that the proposed method improves the validity and code coverage of test cases to a certain extent, and greatly increases the probability of anomaly detection in the protocol implementation

I. Introduction

The IIS (Industrial Internet System) implemented by a variety of automation components to achieve data acquisition, control, monitoring and other functions. A typical industrial Internet communication architecture generally consists of a three-layer structure, from high to low, respectively, the Enterprise Network, Monitoring Network and Control System Network [1][2]. Figure 1 shows a typical Industrial Internet Control System architecture.

The IIS refers to the system composed by the computer equipment and the industrial production control unit, mainly including the SCADA (Supervisory Control and Data Acquisition), the DCS (Distributed Control System), the PCS (Process Control System), the PLC (Programmable Logical Controller), the PCS (Fieldbus Control System) and so on [3].

In the electric IIS, for example, each link, from electricity generation to electricity utilization, has the corresponding electric IIS terminal, such as the PLC, the intelligent substation equipment, the monitoring and control device and other types, for data acquisition, instruction scheduling, remote control, and so on. The transmission of control flow and data flow between the IIS terminal and the main station is realized through Industrial Internet protocols. The program component running in the terminal is responsible for analyzing and processing Industrial Internet protocols.

In recent years, with the rapid development of various emerging information technologies, industrialization and informatization are more closely integrated. More modern information technology has been applied to traditional IIS. Meanwhile, various standardized communication protocols and network switching architectures have been popularized in IIS. Due to the addition of advanced

information technology and communication network technologies (such as Ethernet), the openness of Industrial Internet Control Systems has been greatly enhanced, and it has also exposed it to more security risks. In 2010, Iranian nuclear facilities were attacked by Stuxnet [4], causing the delay of uranium enrichment. Using the vulnerability of Siemens' SIMATIC WinCC/Step7 control system, Stuxnet broke into the SCADA successfully, injected the malicious code into the PCL and hided perfectly to destroy the centrifuge while getting the control power of turbine for the purpose of destruction. As of September 2010, 0.1 million host computers worldwide have been infected by the virus with a strong destructive effect. In 2011, DUQU, Stuxnet's variant [5], forged digital signatures through Microsoft's vulnerability MSII-087 to avoid the detection of security components of system, and then stole various information on system and returned it to the attacker. In 2015, Ukrainian electric power system was attacked by malicious software Black Energy [6] which got the remote access and control power of system and thus caused the crash of power grid's SCADA host system and then an electricity black-out in a large area. In 2017, security manufacturer ESET announced a tool win32/Industroyer attacking electric IIS directly. The tool can cause the blackout from transformer substation by controlling the breaker. In 2018, a lot of warning messages of illegal intranet access appeared on a provincial electric IIS monitoring platform in China. After an analysis, people found the reason was the manufacturer ran and maintained the product server remotely by opening the function of file sharing and thus exposed the system in the public network for a long time, which brought serious hidden dangers. In the past ten years, many security incidents and potential and possible dangers made the security situation of IIS increasingly serious. Figure 2 shows the statistIIS of IIS security incidents in recent years, and the data came from IIS-CERT (The Industrial Internet Systems Cyber Emergency Response Team) [7–13].

During the realization of protocols by Industrial Internet terminal codes, protocol fields are generally credible, but the attacker can control the running of program by changing the data values of the fields using the defects of protocols and then affect the whole system. For example, the skip instruction's destination address parameters generally come from credible data sources in the program rather than externally incredible input such as the incoming data from an Industrial Internet protocol. However, the attacker can overwrite the instruction's destination address through system vulnerability and then control the running process of IIS. Professor Jonathan M. Garibaldi proposed an indistinguishable conceptual framework as a key component of the evaluation of computerized decision support systems. Case studies show that human experts are not perfect, and there are techniques that allow fuzzy systems to simulate human-level performance, including variability. He demonstrated the necessity of "fuzzy intelligence" from two aspects: (a) Fuzzy methodology (in the technical sense of Zade fuzzy sets and systems) is necessary as a knowledge-based system to represent and reason for uncertainty; (b) When evaluating intelligent systems, ambiguity (in a non-technical sense) is required and imperfect performance is accepted[14]. This article is inspired by these ideas and has carried out related research. In this context, to solve the problem of low rate of code coverage caused by the repeated execution of test cases on the same path, starting from the level of system program in the realization of Industrial Internet protocols and in the premise of acquirable source program codes or executable binary file, the paper proposes a method combined with the dynamic multi-modal sensor communication data in the program

for fuzzy processing. The method tacks the program execution of protocol realization, finds the input fields affecting conditional branches through dynamic taint analysis and captures the dependence relationship between conditional branches to guide the generation of test cases pertinently.

Ii. An Industrial Internet Protocol Fuzz Test Method Based On Dynamic Analysis

In view of the serious damages caused by IIS' vulnerability, researchers have proposed many vulnerability discovery technologies, such as the dynamic analysis, the symbolic execution, the fuzz test and so on [15][16]. Comparing with other technologies, the fuzz test requires only a little knowledge of target, but can expand to large application program easily with good reusability, and thus has become the most popular vulnerability discovery solution currently, especially in the IIS.

With regard to network protocols, the most representative fuzzifier is SPIKE [17]. Its principle is describing the protocol as a block sequence model and making the data variation in blocks, and then partitioning the message's data structure and making statistIIS of field length after the variation automatically, thus improving the effectiveness of test cases significantly, but the fuzzifier shows an inadequate descriptive power for the constrained relationship in protocol messages. Later, Sulley [18] and Peach [19] extended the data model based on SPIKE and added more descriptions on the dependence relationship between data blocks. In order to provide a more flexible and accurate fuzzy framework, AFL[20] tracked the path coverage of each input through the lightweight instrumentation in source program, and allocated ID randomly to the basic blocks on the path using the Hash mechanism to determine whether where is any new path generated, and then used the input generating a new path as the seed. Combining with the detailed information in the program, the method improves the code coverage rate, but the Hash method may have collision cases easily and thus cause the problem of false negatives even though the input has reached a new path. Gan et al. proposed CollAFL [21] which allocated the ID value to each basic block using the greedy algorithm and other methods to ensure the Hash value is difference in each side and thus avoided Hash collision and realized a more accurate judgment of path coverage. Besides, the method uses the number of neighbor branches of path as the weight to sort seeds and prompts the fuzzifier to probe the paths haven't been reached. Sanjay et al. proposed VUzzer [22]. The method requires no prior knowledge of application program or input format, and mainly uses the control flow and data flow characteristIIS of static and dynamic analyses to assist in variation and selecting seed files, and prompts the program to execute on a deeper level by giving the specific weight to each basic block.

With regard to the Industrial Internet protocol, most research achievements were realized by improving and extending existing network protocol fuzz test methods or tools. For instance, Roland et al. proposed ProFuzz [23] specific for the Profinet protocol stack. ProFuzz is mainly based on the fuzz test tools developed by Scapy fuzzer [24], and also contains Sullley's fuzzer module, supporting five types including cyclic real-time execution, device discovery, configuration acquisition, application request and accurate time control protocol. With regard to Wurldtech Company's BlackPeer test framework [25], its key to constructing anomalous data is, in the premise of given initial sample data, defining protocol messages

with the extended Backus-Naur form and generating corresponding test data grammar and then realizing the variation of protocol data units using interactive semantIIS. On the basis, Yafeng Zhang et al. [9] improved the BNF grammar. They parsed the industrial protocol samples into the variation tree by introducing a tree-shape structure, and traversed all nodes in the tree for the purpose of variation. McCorkle et al. [26] made a fuzz test to the upper computer software of IIS, constructing anomalous data by making the variation operation to response messages and keeping the communication state of protocol by forging check codes and count values. SecuriTeam added the DNP3 protocol into the test tool beSTORM [27] and found the vulnerability of denial of service attack specific for DNP3 during the fuzz test to Wireshark. The Mu suite launched by Mu Dynamic Company [28] is applicable to protocols of IEC 61850, Modbus/TCP and DNP3, which constructs abnormal message data with the structured grammar analysis method and can also extend the Industrial Internet protocols with unknown specifications using the additional function Studio Fuzz. Lzfuzz [29] is used for the specific DCADA protocols with unknown grammar, of which the basic idea is deducing message tokens using the Lempel-Ziv compression algorithm and, combined with token information, making the variation processing to samples, but the method can hardly be used for the protocols with complicated input grammar.

From the research above, we summarize the problems of fuzz test methods applying to Industrial Internet protocols currently as follows. (a) The low code coverage. Many bugs can be triggered only in the case of a large path coverage rate. Although some methods have used the dynamic multi-modal sensor communication data processed in the program, it is not reflected in the generation of test cases directly, causing that most cases are executed repeatedly on the same paths as input data and only cover a few paths. (b) No unified description model. Protocols, even of the same type, have different forms of messages, but current methods require building different data models, thus increasing the construction workload of model. (c) Insufficient pertinence of test cases. It’s hard for test cases to pass the program verification, thus causing too many invalid tests. The design of variation strategy without taking the characteristIIS of Industrial Internet protocols into full consideration may cause the redundancy of case in the case of a lot of sample data and then affect test efficiency.

A. Dynamic Taint Analysis

The dynamic taint analysis proposed by J. Newsome [30] is a method tracking information flow during the running of program, i.e., tracking the transmission of data slots to be analyzed in the system and gaining target program’s detailed processing process for the data. In the method proposed, we use the technology to get the dynamic multi-modal sensor communication data in the program to provide the basis for the further fuzz test.

The dynamic taint analysis method involves two parts: the taint data identification and the taint transmission path monitoring [31]. The core of taint data identification is defining taint data. If the data source is suspect, the data generated by it are taint data, which, as the input, shall be tracked and analyzed in the running process of binary code, and then the message data constructed in the fuzz test can be considered as suspect. During the running of program, the transmission of taint data is completed

through instructions and taint data may participate in the operation as the source operands in the arithmetical operation instruction or the parameters of data movement instruction, of which the operation output is generally related to taint tracking data and thus should be identified as the taint attribute. Figure 3 shows a simple example of dynamic analysis process. In the figure, a_1 and a_2 are taint data; the arrow means the operation execution; the leading end represents the source operand; the tail end represents the operation output. During the running of program, variables $V_1 \sim V_8$ are affected by taint data a_1 and a_2. If A, as the set of taint source of each variable, is empty, we can consider that the variable is not tainted. From the figure we can see that the set of taint source of V_8 which is the last output is the union set of taint source sets of operation operands V_3 and V_6.

Using the dynamic taint method, we mainly aim to analyze the data flow and the control flow, also known as the explicit flow and the implicit flow, during the execution of binary codes. The former refers to data dependence relationship, i.e. variable V_1's taint information is sent to V_2 directly through the assignment or arithmetic operation, as shown in Figure 3; the latter corresponds to the control & dependence relationship of conditional branches, i.e. variable V_1's taint information is sent to V_2 indirectly through the associated condition expression. For the purpose of easy understanding, we make a simple analysis using the example of sample code shown in Figure 4.

In the example above, the function of code implementation experiences the input of variable x, the generation of message msg and the submission through post. In the process, x is identified as the taint datum, and according to the analysis method of data flow, msg is assigned as constant 'a' or 'b' directly. The constant won't be tainted, so msg is identified as the untainted attribute. However, the value of msg also depends on whether conditional branches x=='a' and x=='b' are true or false, i.e. msg and x have a control & dependence relationship, which belongs to the implicit taint of flow. In practical applications, such codes have a security risk when used for the communication realization of network protocols.

When msg is submitted, the attacker may intercept msg and then deduce the value of input x, so msg is definitely a taint datum and thus should be tacked and monitored. There will be false negatives without considering the control & dependence relationship of msg. On the contrary, url's value is independent of branches L3 and L7, so there is no harm even if it is identified as the untainted datum.

B. Method and Principle

According to the Industrial Internet protocol realization program, the section gets related dynamic multi-modal sensor communication data through dynamic taint analysis to guide the generation of test cases. Figure 5 shows the specific test process of the method. The method proposed uses many protocol messages as the input to avoid the problem of insufficient test cases caused by single sample datum.

Definition 1 The dynamic interactive fields: In the given program execution, then for conditional branch x_i (the i^{th} execution of conditional branch x), there is $DIF(x_i)=\{F_j|F_j$ is the protocol field affecting the execution of $x_i\}$ in which $DIF(x_i)$ is the set of protocol fields.

Definition 2 The dependence & control relationship: In the given program execution, then if there is a conditional branch y_j deciding whether to execute x_i, we can say x_i depends on y_j's dynamic control and express it as $CDC(x_i)=\{y_j,T|F\}$ in which $CDC(x_i)$ is the set of branches meeting the condition, and Constraint(x_i) represents the constraint of conditional branch x_i.

Definition 3 Dynamic control flow diagram: For each program input, and its execution path can be expressed with the dynamic control flow diagram, in which conditional branch x_i, is the node, DIF(x_i) is the dynamic interactive field of node, and $CDC(x_i)$ is the side representing the dependence relationship of conditional branches.

We define the input protocol fields affecting conditional branches through the dynamic taint analysis, make the taint identification processing to each protocol input field and track the tainted data flow in the execution of program. With regard to the dependence & control relationship of program, we capture it with the algorithm proposed in literature document [32]. It should be noted that for Industrial Internet protocols, the fields generally have the checksum (like the CRC) which will cause the deluge of taint data if transmitted in control information flow. Because of all fields used as checksums, most conditional branches will be identified as taint data, in which case, it's hard to directly find the specific field affecting conditional branches. Besides, due to the dependence & control relationship, the fuzzy method proposed considers the transmission taint in control flow indirectly. Therefore, when using the dynamic taint analysis method, we only focus on the transmission taint in the data flow rather than tracking the transmission taint in the control flow.

Algorithm 1 introduces the fuzzy method proposed in details. The main function dynamicFuzz can execute program P, Protocol G and protocol message I as the input, to process much protocol input, and the output is the abnormal information triggered. Lines 1~2 of algorithm initialize the data structure to be stored. The test cases generated which will also be used as the new input are put in the queue for storage for convenience of recursion execution. Line 5 combined with two parameters, program and protocol input, gets the sets of $DIF(x_i)$ and $CDC(x_i)$ of each conditional branch in each program using the dynamic taint analysis method, and then constructs the corresponding dynamic control flow diagram using $DIF(x_i)$ as the node and $CDC(x_i)$ as the side. Line 6 makes the fuzzy operation from the start point of executable path in the control flow diagram. Line 8 deduces Constraint(x_i), the corresponding constraint condition of node x_i, as the parameter of test case generation algorithm.

Line 16 of algorithm means that after the generation of test case, we need to use the test case as the new input to replace the value of protocol field in set $DIF(x_i)$ of original input and modify all fields related to $Constraint(x_i)$ to meet the constraint condition. To ensure the test case will be executed on a deeper level in the program and improve the probability of finding new execution paths, the rest of protocol fields which are unrelated to $DIF(x_i)$ and $Constraint(x_i)$ should also be kept valid according to protocol grammar. For instance, in the protocol input, the values of two fields start address and end address shall change from original 1 and 2 into 3 and 6. Even if there is no constraint condition, the corresponding fields of

address data shall be regenerated based on the original normal size (increasing from 1 to 3), but if the two address change into the invalid test cases of 6 and 3 after the fuzzification (start address < end address), the fields unrelated to $DIF(x_i)$ and $Constraint(x_i)$ still keep the normal values.

Lines 17~20 monitors whether there is any abnormal state such as the crash or memory leak in the execution of test case. Although the test case is used as the new input, in consideration of the expenses caused by program execution, the method proposed doesn't get the dynamic multi-modal sensor communication data of each conditional branch repeatedly. In the execution of test case, Lines 21~24 store the code paths traversed, put the conditional branches unparsed into the input queue and then decide the priority ranking according to the quantity of new branches. Line 25 of algorithm stores the dynamic multi-modal sensor communication data sequence of each node for lines 9~13 to judge whether current node generates the test case. When the list has a dynamic multi-modal sensor communication data sequence and the test case generated which correspond to current node x_i's $DIF(x_i)$ and $Constraint(x_i)$ respectively, then the algorithm will skip the test case generation of the node directly. Line 29 of algorithm stores the dynamic control flow diagrams as the paths probed in the queue after completing the code paths of program.

Algorithm 1. A Fuzz test Algorithm Combined with Dynamic multi-modal sensor communication data

Input: Program *P*, Protocol Grammar *G* and Protocol Message *I*
Output: Abnormal Information
Function dynamicFuzz (*P*, *G*, *I*)

```
1. probedPath,inputQueue, checklist ← empty()
2. inputQueue.push(I)
3. while inputQueue.notEmpty() do
4. input ← inputQueue.pop()
5. dcfg ← executionAnalysis ( P, I)
6.  node ← dcfg.start(probedPath)
7.  while node ≠ NULL do
8.  c ← dcfg.getConstraint (node.CDC)
9.  if checklist.find(node.DIF, c) &&
10. node.true, node.false ≠ NULL then
11. node ← dcfg.next()
12. continue
13. end
14. tcList ← makeTestCases (node.DIF, c, G)
15. for each tc in tcList do
16. input'← makeInput (input, tc.value, c)
17. res ← executionMonitor( P, input')
18.  if (res.execption) then
19. return getExceptionInfo(res)
20. end
21. if(probedPath.isNew(res.pathInfo)) then
22. inputQueue.push(input')
23. end
24. end
25. checklist.add(node.DIF, c)
26. node ← dcfg.next()
27. end
28. end
29. probedPath.add(dcfg)
```

To reflect the dynamic multi-modal sensor communication data extracted from the program into the construction of test case directly, the method proposed focuses on the generation technology assisted with the variation, and guides the generation of test case grammar specific for node x_i by combining with

corresponding dynamic multi-modal sensor communication data. Algorithm 2 describes the specific process. The test case generation function makeTestCases considers protocol field *fields*, i.e. the element in set $DIF(x_i)$ in Definition 1, rule constraint *c* and protocol *G* as the input, and the output is the set of test cases *tcList*. The key to the function is generating test cases by acquiring the efficient grammar of each node and the reversed grammar. Line 2 has only one valid grammar for protocol fields, i.e. deducing the grammar of each protocol field in the constraint condition according to parameters as the valid grammar of the node. The feasibility is the valid grammar applied to node x_i definitely is the subset of protocol grammar G, because according to Definition 3.2, i.e. in the constraint condition of $Constraint(x_i)$, the valid grammar can apply to all the protocol fields in $DIF(x_i)$. Line 3 means when the protocol field has much valid grammar, constructing much grammar through the combinations of the valid grammar and storing the grammar in the set of test cases. For instance, for node xi, there is $DIF(x_i)=\{F_a,F_b\}$, and the valid grammar deduced in $Constraint(x_i)$ is F_a=(0|1), F_b=2, and then the grammar combined is (F_a=0, F_b=2) and (F_a=1, F_b=2). Next, Lines 4~9 of algorithm reverse the valid grammar corresponding to each filed of valid grammar in the premise of meeting constraint condition $Constraint(x_i)$, and then combine them with other fields' valid grammar for fuzzified grammar and add the grammar to the set of test cases.

When there is no applicable test grammar found in $DIF(x_i)$, the case data shall be generated through variation with methods like random bit flip, extreme substitution, boundary value substitution, and so on.

Algorithm 2. A Test Case Generation Algorithm Combined with Dynamic multi-modal sensor communication data

Input: Protocol Field *fields*, Rule Constraint *c* and Protocol Grammar *G*
Output: Set of Test Cases *tcList*
Function makeTestCases(fields, c, G)

```
1. tcList ← empty()
2. validGrams ← extractGrams(fields, c, G)
3. tcList ← combination (validGrams)  c
4. for each g in validGrams do
5.   fg ← ~g  c
6.  fuzzyGram ← fg  (other g'in validGrams)
7.  tcList.add(fuzzyGram);
8. end
9. return tcList
```

Iii. Methods And Experimental

We use the Modbus protocol as an example. Figure 6 shows the simplified format of Modbus protocol. The fields include the protocol identity, the length of information, the field of function code, the start address, the end address, the data field determined according to function code and the CRC (Cyclic Redundancy Check). The figures in brackets represent the sizes of fields. The values of CRC calculation and other checksum algorithms as the mechanisms completing protocol communication are generally imbedded into protocol specifications to identify potential broken data. Although the verification mechanism of Modbus/TCP is realized in the transmission frame of TCP and the protocol formats don't include the CRC, the CRC verification generally exists in Modbus RTU and other Industrial Internet protocols. If the CRC received by the program fails to match the CRC obtained through calculation, the data of corresponding case may be ignored directly. It is a very important mechanism to ensure security

and functionality, but if CRC's value is not updated when protocol fields change, the fuzz test shall be blocked, so the CRC is also considered as an important field.

Figure 7 describes some program codes processing protocol realization. Line 3 conditional branch of program checks whether the protocol identity meets specifications and verifies the validity of input message data. The protocol messages failing to meet specifications shall be abandoned directly without any processing. Line 8 is the CRC. Similarly, the messages failing to pass the check shall also be abandoned directly. If the conditional branches above are passed, the next operation is reading the function code. The example program only offers two typical function codes of Modbus protocol: reading and writing, i.e., function code 0x05 supported represents the writing operation, and 0x01 represents the reading operation. The example code has two bugs, and these anomalies shall be triggered only in the case of meeting specific conditional branches. Line 22 describes the typical heap overflow bug which means the program fails to check whether the size of data from the start address to the end address is consistent with the size of actual message and thus causes the problem of heap overflow when the length of data exceeds the size of actual data. Lines 24~26 are the bugs embedded deliberately, which shall be triggered only if a specific value (KEY) is written into a specific address (TRIG_POINT). We can see clearly that for the test case input, the bug may be triggered only through the protocol identity and CRC. It is difficult for the fuzz test method based on random variation which doesn't know the grammar actually executed by the target protocol in the program and thus can't reach the conditional branches may trigger the anomaly and also causes a lot of invalid test data. According to the algorithm above, we analyze the protocol input format in Figure 6 and the protocol program example in Figure 7. For the convenience of description, we call the protocol fields as *I*, *L*, *F*, *S*, *E*, *D* and *C* for short.

Realization

Figure 8 shows the control flow diagram and execution paths of the first input. It is a dynamic control flow diagram constructed through a valid input executing the writing function, of which the right half part shows the paths actually covered by the input during the running of example program and the dotted part shows the new execution paths discovered in each node. According to Definition 3, in the figure, the start node 3_1 represents the first execution of the conditional branch corresponding to the 3rd line of example code; the elements in braces represent $DIF(3_1)$, the field affecting the 3rd line conditional branch; the side with arrow represents the dependence & control relationship $CDC(3_1)$. The control flow diagram shown in Figure 9 is the test case generated by choosing the combinations of function code 0x05 and *n* data fields in the case node 24_1 is true in the first input execution, and is considered as the second input.

Suppose the start node 3_1 doesn't depend on any node, considers protocol identity field *I* as the dynamic interactive field, and has the only true value of 0x0000, and then get the corresponding two test grammar after fuzzification with the algorithm. Node 3_1 actually is a conditional branch to identify whether the field is true or false according to the protocol, so according to each test grammar, it's easy to deduce that the constraint condition making node 3_1 true is . Briefly speaking, the condition making the conditional

branch of line 3 executable is , while the constraint making the conditional branch false is I=0x0000. In this case, for the next node 8_1, according to its $CDC(8_1)$, deduce $Constraint(8_1)$ is the constraint making 3_1's conditional branch false, i.e. I=0x0000. The cyclic redundancy filed, as the only grammar check field applicable to the whole field, generates two test grammar I=0x000∧crc(I, L, F, S, E, D)=C and I=0x000∧crc(I, L, F, S, E, D)≠C in a similar way. On the basis, go on getting the constraint condition of node 8_1, and then, in the similar manner, solve each node.

It should be noted that according to the algorithm description, the test grammar generated in node 13_1 has decided whether node 14_1 is true or false, so there is no corresponding test grammar generated in node 141. Similarly, skip nodes 21_1, 21_2, 22_2, 24_2 and 21_3. According to algorithm 1, when the test grammar is generated, the dynamic flow diagram is stored into the queue as the probed path. In this case, if there is the probed path, when searching paths in the control flow diagram, we can get the partition node generating different paths by comparing the paths and consider the node as a start node. All nodes before the node have the same dynamic multi-modal sensor communication data and thus can be extracted directly to avoid the repeated generation of test cases. Table 1 gives an example of the test case generation grammar of the first input obtained with algorithm 2.

Table I Test Case Generation Grammar Rules

Node x_i	DIF(x_i)	CDC(x_i)	Constraint(x_i)	Valid Grammar	Grammar after Fuzzification	Test Grammar
3_1	I	ø	ø	I=0x0000	I≠0x0000	I=0x0000 I≠0x0000
8_1	I~C	(3_1,F)	I=0x0000	$Crc(I\sim D)=C$	$Crc(I\sim D)\neq C$	$Crc(I\sim D)=C$ $Crc(I\sim D)\neq C$
13_1	F	(8_1,F)	I=0x0000 $crc(I\sim D)$=C	F=0x01	F≠0x01	F=0x01 F≠0x01
14_1	F	(13_1,F)	I=0x0000 $crc(I\sim D)$=C F≠0x01	skip	skip	skip
17_1	S, E	(14_1,T)	I=0x0000 $crc(I\sim D)$=C F=0x05	S≤E	S>E	S≤E S>E
21_1	S, E	(17_1,F)	I=0x0000 $crc(I\sim D)$=C F=0x05 S≤E	skip	skip	skip
21_1	L, S, E	(21_1,T)	I=0x0000 $crc(I\sim D)$=C F_3={1}, F_4≤F_5	Input size is valid	Input size is valid	Input size is valid Input size is valid

We make an experiment using the most popular Modbus/TCP protocol as an example. Figure 10 shows the test environment created.

Table 2 gives the equipment involved in the figure.The experiment is made with Ubuntu 14.04, so we choose the open source library libmodbus to realize the Modbus/TCP protocol communication in Linux

system, as shown in Figure 11. The fuzzy method proposed is realized with Symfuzz [33], a tool developed based on BAP [34] open source binary system analysis framework, which can convert executable files into the intermediary language applicable to program analysis, and combined with PIN to execute dynamic binary system instrumentation to the target program to get the dynamic interactive fields and dependence & control relationship required by the method to guide the generation of subsequent test case. We choose AFL-fuzzer as the fuzzy tool in the comparison experiment.

Table II Network Equipment In The Test Environment

Equipment	Quantity	Use
Moduus Main Station	1	The Modbus client side to realize writing and reading operation
Moduus Slave Station	1	The Modbus server to respond to data request
Fuzzer	1	The fuzz test tool to construct anomalous data

The Modbus slave station (libmodbus server) waits for the request messages from other main stations. In the experiment, we set that the main station messages send 5, 10 and 15 request messages, as test input, to the slave station, respectively, and makes the fuzzy processing to Modbus slave station program and monitors whether the target program has any anomaly.

To evaluate the performance of the fuzz test method proposed, we make a comparison in terms of the quantity of test cases, the total number of execution paths, the code coverage rate and the test time with the same number of samples.

The experiment makes the statistIIS on the number of test cases generated using two fuzz test methods with 5, 10 and 15 sample data. The number of test cases refers to the total number of samples generated after the crash of program or the complete execution of samples. Figure 12 shows that with the increase in the number of samples, the test data generated with the method proposed are significantly less than the test data generated with AFL-fuzzer, because the method proposed constructs test cases using the generation-based technology in which case the test cases generated are definitely less than those of AFL-fuzzer using the variation strategy. Besides, we can see the number of test cases generated by AFL-fuzzer grows steadily as the number of samples increases, because Afl-fuzzer prunes input samples in consideration of that users may offer low-quality initial samples and thus cause the possible data redundancy in some types of variations. Although depending more on the number of samples, the method proposed adopts a more pertinent generation strategy [35-47].

Samples

Figure 13 shows the coverage rate of code. The calculation principle is that the instrumentation can help capture the coverage rate of branch and detect the rough hit count of branch execution. The code coverage rate doesn't necessarily have any connection with the probability of finding anomalies, but undoubtedly, for a test case, an execution path failing to reach the program's conditional branch on a

deep level certainly will not trigger any potential anomaly. So, we can see that the method proposed realizes a higher rate of code coverage compared with the AFL-fuzzer.

Table 3 shows the statistIIS when the target program crashes. The process of guiding test case generation by combining with program dynamic multi-modal sensor communication data pays the price of test time for solving conditional branch constraint and generating test grammar, compared with the variation strategy. However, it significantly increases the number of execution paths and the code coverage rate, indicating in the running of program, the more execution paths are found by the conditional branches traversed, the bigger the probability that test cases reach the deep level and trigger anomalies is, and the stronger the pertinence is. Therefore, in the case the program has anomalies, although scarifying some test time, the method proposed is superior to the AFL-fuzzer in terms of realizing Modbus-TCP protocol test.

Table III Comparison Results Of Experiment

Test Method	Number of Test Cases	Number of Execution Paths	Code Coverage Rate	Test Time /h	Anomaly
Method proposed	16247	2896	50.90%	4.19	1
AFL-fuzzer	38652	1057	41.85%	3.35	1

Iv. Conclusion

The work proposes a fuzzy processing method combined with dynamic multi-modal sensor communication data from the level of system program in Industrial Internet protocol realization. The paper first expounds the theory of dynamic taint analysis and then gives the definition of dynamic multi-modal sensor communication data required and, proposes the fuzz test method combined with dynamic multi-modal sensor communication data. The method proposed tracks program execution, finds the input fields affecting conditional branches through the dynamic taint analysis, and captures the dependence relationship of conditional branches to guide test case grammar generation pertinently, thus increasing the opportunity of executing codes on the deep level. The results of comparison experiment prove that the method improves the validity of test cases and the coverage rate of codes to some extent, and also increases the probability of finding the anomalies in protocol realization greatly.

Results And Discussion

Aiming at the problems of incomplete test coverage and low processing efficiency of the existing fuzz testing technology based on program feedback, this paper proposes a method of fuzzy processing combined with dynamic information. The method in this paper tracks the execution of the program, finds out the input fields that affect the execution of the conditional branch through dynamic pollution analysis, and captures the dependencies between the conditional branches to guide the generation of test cases in a targeted manner. The method in this article can increase the chance of executing deep code, to a certain extent, improve the effectiveness of test cases and code coverage, and also increase the probability of finding anomalies in the implementation of the protocol.

However, follow-up research needs to deepen around the following areas:

1.The method in this paper is a dynamic and targeted test from the perspective of the program under test. Dynamic analysis of the program also causes a large cost. So, in the next step, we consider combining the static analysis technology of the program to reduce the burden of dynamic analysis, and at the same time use the obtained dynamic information to adjust the position of the blur and the combination strategy.

2.The industrial control protocol model in this article only describes the data format level. In actual production, it is necessary to consider that the protocol has state implementation. Therefore, the description of the industrial control protocol state machine needs to be studied in depth to establish the association between the protocol format and the protocol state transition.

Declarations

Acknowledgements

We want to thank the authors of the literature cited in this paper for contributing useful ideas to this study.

Availability of data and materials

Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

Authors' contributions

Qianmu Li, Shunmei Meng, Hanrui Zhang, Yaozong Liu, Haiyuan Shen and Huaqiu Long have written this paper and have done the research which supports it. Qianmu Li, Shunmei Meng, Hanrui Zhang has collaborated in the conception, research and design of the paper. All authors read and approved the final manuscript.

Author information

Qianmu Li received the BSc and PhD degrees from Nanjing University of Science and Technology, China, in 2001 and 2005, respectively. He is currently a full professor with the School of Cyber Science and Engineering, Nanjing University of Science and Technology, China. His research interests include information security and data mining. He received the China Network and Information Security Outstanding Talent Award in 2016, and Education Ministry Science and Technology Awards in 2012.

Shunmei Meng received her PhD degree in Department of Computer Science and Technology from Nanjing University, China, in 2016. Now, she is an assistant professor of School of Computer Science and Technology, Nanjing University of Science and Technology, Nanjing, China. She has published papers in

international journals and international conferences such as TPDS, ICWS, and ICSOC. Her research interests include recommender systems, service computing, and cloud computing.

Hanrui Zhang received the BSc. degree from Nanjing University of Science and Technology, China, in 2017. She is now a Ph.D. student at Nanjing University of Science and Technology. Her research interests include data mining and network security.

Yaozong Liu received the Ph.D. degree from the Nanjing University of Science and Technology, China, in 2016. He is currently a Lecturer with the Intelligent Manufacturing Department, Wuyi University, China. His research interests include data mining and network security.

Haiyuan Shen was born in 1984. Master's degree. After graduating with a master's degree in 2009, he became a software engineer. He also serves as the director of Zhongtian Software. He works in Jiangsu Zhongtian Technology Co., Ltd. His research interests include service computing and cloud computing.

Huaqiu Long received his BSc degree form Intelligent Manufacturing Department, Wuyi University, in 2019. He currently works in the Cyber Security Lab of Wuyi University. In 2019, he was admitted to a part-time master's degree. His research interests include information security, computing system management, and data mining.

Funding

This work was supported in part by the 2019 Industrial Internet Innovation and Development Project from Ministry of Industry and Information Technology of China, 2018 Jiangsu Province Major Technical Research Project "Information Security Simulation System", Fundamental Research Funds for the Central Universities (30918012204), Military Common Information System Equipment Pre-research Special Technology Project (315075701).

Competing interests

The authors declare that there is no conflict of interest regarding the publication of this manuscript.

References

1. Yong Peng, Changqing Jiang, Feng Xie et al., "IIS Information Security Research Progress," Journal of Tsinghua University: Natural Science Edition, vol.10, pp. 1396-1408, 2012.
2. Ericd, Knapp et al., Industrial Network Security: Smart Power Grids, SCADA and other IIS Key Infrastructures, CA: NDIP, 2014, pp.18-26.
3. A. Stouffer, Make Lowercase et al., "SP 800-82. Guide to Industrial Internet Systems (IIS) Security: Supervisory Control and Data Acquisition (SCADA) systems, Distributed Control Systems (DCS), and other control system configurations such as Programmable Logic Controllers (PLC)," 2011.

4. Yao Dong, Xiaohua Yang & Shubin Jin, “An Analysis on Stuxnet’s Influence on IIS’ Security,” Report of Times (Academic Edition), no.1, pp. 64, 2013.
5. Chien E, Omurchu L& Falliere N, “W32.Duqu: the precursor to the next stuxnet,” in Usenix Conference on Large-scale Exploits & Emergent Threats, San Jose, CA, 2012.
6. RAVAL S, “Black Energy a threat to Industrial Internet Systems network security,” International Journal of Advance Research in Engineering, Science Technology(IJAREST), vol. 2, pp. 31-34, 2015
7. IIS-CERT. Information Products [EB/OL], https://IIS-cert.us-cert.gov/, 2018.
8. China National Vulnerability Database. Vulnerability of Industrial Internet Industry [EB/OL], http://IIS.cnvd.org.cn/, 2018.
9. Yafeng Zhang, Zheng Hong, Lifa Wu, et al., “Paradigmatic-grammar-based Industrial Protocol Fuzzing Test Technology,” Application Research of Computer, vol.33, 2016.
10. S Wan, M Li, G Liu, C Wang, Recent advances in consensus protocols for blockchain: a survey. Wireless Networks, 1-15, 2019.
11. Ndiaye M A A, Petin J F, Camerini J, et al., “Performance assessment of Industrial Internet system during pre-sales uncertain context using automatic Colored Petri Nets model generation” in International Conference on Control, Belfast, 2016.
12. Banerjee S, Großmann I D., “An Electronic Device Description Language based approach for communication with dbms and file system in an industrial automation scenario,” in IEEE International Conference on Emerging Technologies & Factory Automation, 2016.
13. Chang Luo, “Research on IIS Information Security Protection System’s Application in Electric Power System,” Mechanical & Electrical Engineering Technology, vol.12, 2016.
14. Jonathan M. Garibaldi, "The Need for Fuzzy AI," IEEE/CAA J. Autom. Sinica, vol. 6, no. 3, pp. 610-622, 2019.
15. Liu B, Shi L, Cai Z, et al., “Software Vulnerability Discovery Techniques: A Survey” in Fourth International Conference on Multimedia Information Networking & Security. IEEE, 2013.
16. Hantao Sun, “The Research on Software Security Vulnerability Mining Technology Based on Fuzz test,” Electronic Technology & Software Engineering, vol.13, pp.83-84, 2016.
17. Aitel D., An introduction to SPIKE, the fuzzer creation kit [EB/OL], http://www.blackhat.com/presentations/bh-usa-02/bh-us-02-aitel-spike.ppt, Accessed, 2014-01-06
18. Devarajan G.Unraveling, “SCADA protocols: using Sulleyfuzzer [EB/OL],”http://www.defcon.org/html/defcon-15/dc15speakers.html, 2015-06-21.
19. [EB/OL]. http://www.peachFuzzer.com, 2015-07-21
20. Zalewski, American fuzzy lop [EB/OL]. http://lcamtuf.coredump.cx /afl/ , 2017-12-25.
21. Gan S, Zhang C, Qin X, et al., “CollAFL: Path Sensitive Fuzzing,” in 2018 IEEE Symposium on Security and Privacy (SP), SanFransisco, CA, USA: IEEE Computer Society, 2018, pp.660-677
22. Rawat S, Jain V, Kumar A, et al., “Vuzzer: Application-aware evolutionary fuzzing,” in Proceedings of the Network and Distributed System Security Symposium (NDSS), SanDiego, CA, USA: Internet

Society, 2017
23. Koch R. Profuzz [EB/OL]. https://github.com/HSASec /ProFuzz, 2015-06-21.
24. Philippe Biondi. Scapy, python interactive packet manipulation framework [EB/OL] https: //www.secdev.org /projects /scapy /,2012
25. Byres E J , Hoffman D , Kube N ., " On shaky ground-a study of security vulnerabilities in control protocols ," in 5th American Nuclear Society International Topical Meeting on Nuclear Plant Instrumentation, Controls, and Human Machine Interface Technology, American Nuclear Society, 2006.
26. Michael Toecker: Response fuzzing [EB/OL]. http: //www.digitalbond.com/blog /response-Fuzzing /, 2013-05-08
27. Blackops: Pattern recognition, presented at the Black-HatUSA conference [EB/OL]. http: //www.slideshare.net /dakami /dmk-blackops, 2006, 2013
28. DynamIIS M.Mu test suite [EB /OL]. http://www.mudynamIIS.com /products/mu-test-suite.html, 2015-06-21.
29. Shapiro R, Bratus S, Rogers E, et al., "Identifying Vulnerabilities in SCADA Systems via Fuzz-Testing," in Critical Infrastructure Protection V-ifip Wg 1110 International Conference on Critical Infrastructure Protection, 2011.
30. Newsome, J., & Song, D. X., "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software, NDSS, vol. 5, pp. 3-4, 2005.
31. Ligen Chen, Shengli Liu, Da Xiao, et al., "A Cisco IOS Heuristic Fuzz test Method," Computer Engineering, vol. 40, pp. 68-73, 2014.
32. Xin B, Zhang X., "Efficient online detection of dynamic control dependence," Proceedings of the 2007 international symposium on Software testing and analysis, 185-195, 2007.
33. Cha S K, Woo M, Brumley D, "Program-Adaptive Mutational Fuzzing," in 2015 IEEE Symposium on Security and Privacy, San Jose, CA, 2015.
34. Brumley D , Jager I , Avgerinos T , et al., " BAP: A Binary Analysis Platform," in International Conference on Computer Aided Verification, Berlin, Heidelberg, 2011, pp. 463-469.
35. Xiaoxiao Chi, Chao Yan, Hao Wang, Wajid Rafique, Lianyong Qi. Amplified LSH-based Recommender Systems with Privacy Protection. Concurrency and Computation: Practice and Experience, 2020. DOI: 10.1002/CPE.5681
36. Lianyong Qi, Xuyun Zhang, Wanchun Dou and Qiang Ni. A Distributed Locality-Sensitive Hashing based Approach for Cloud Service Recommendation from Multi-Source Data. IEEE Journal on Selected Areas in Communications, 35(11): 2616-2624, 2017.
37. Hanwen Liu, Huaizhen Kou, Chao Yan, Lianyong Qi. Link prediction in Paper Citation Network to Construct Paper Correlated Graph. EURASIP Journal on Wireless Communications and Networking, 2019.DOI: 10.1186/s13638-019-1561-7.

38. Lianyong Qi, Xuyun Zhang, Wanchun Dou, Chunhua Hu, Chi Yang, Jinjun Chen. A Two-stage Locality-Sensitive Hashing Based Approach for Privacy-Preserving Mobile Service Recommendation in Cross-Platform Edge Environment. Future Generation Computer Systems, 88: 636-643, 2018
39. S Wan, S Goudos, Faster R-CNN for Multi-class Fruit Detection using a Robotic Vision System, Computer Networks, 107036, 2019
40. Wenwen Gong, Lianyong Qi, Yanwei Xu. Privacy-aware Multidimensional Mobile Service Quality Prediction and Recommendation in Distributed Fog Environment. Wireless Communications and Mobile Computing, vol. 2018, Article ID 3075849, 8 pages, 2018.
41. Wan, S., Li, X., Xue, Y. et al. Efficient computation offloading for Internet of Vehicles in edge computing-assisted 5G networks. J Supercomput (2019). https://doi.org/10.1007/s11227-019-03011-4
42. Lianyong Qi, Xuyun Zhang, Shancang Li, Shaohua Wan, Yiping Wen, Wenwen Gong. Spatial-Temporal Data-driven Service Recommendation with Privacy-preservation. Information Sciences, 2019. DOI: 10.1016/j.ins.2019.11.021.
43. S Wan, L Qi, X Xu, C Tong, Z Gu. Deep Learning Models for Real-time Human Activity Recognition with Smartphones, Mobile Networks and Applications, 1-13, 2019.
44. S Ding, S Qu, Y Xi, S Wan. Stimulus-driven and concept-driven analysis for image caption generation, Neurocomputing, 2019
45. Hou, Q. Li, S. Meng, Z. Ni, Y. Chen and Y. Liu, "DPRF: A Differential Privacy Protection Random Forest," in IEEE Access, vol. 7, pp. 130707-130720, 2019. DOI: 10.1109/ACCESS.2019.2939891
46. Hou, J., Li, Q., Cui, S. et al. low-cohesion differential privacy protection for industrial Internet. J Supercomputing (2020). vol. 7, pp. 1-23, DOI:10.1007/s11227-019-03122-y
47. J Hou, Q. Li, R. Tan, S. Meng, H. Zhang and S. Zhang, "An Intrusion Tracking Watermarking Scheme," in IEEE Access, vol. 7, pp. 141438-141455, 2019. DOI: 10.1109/ACCESS.2019.2943493

Figures

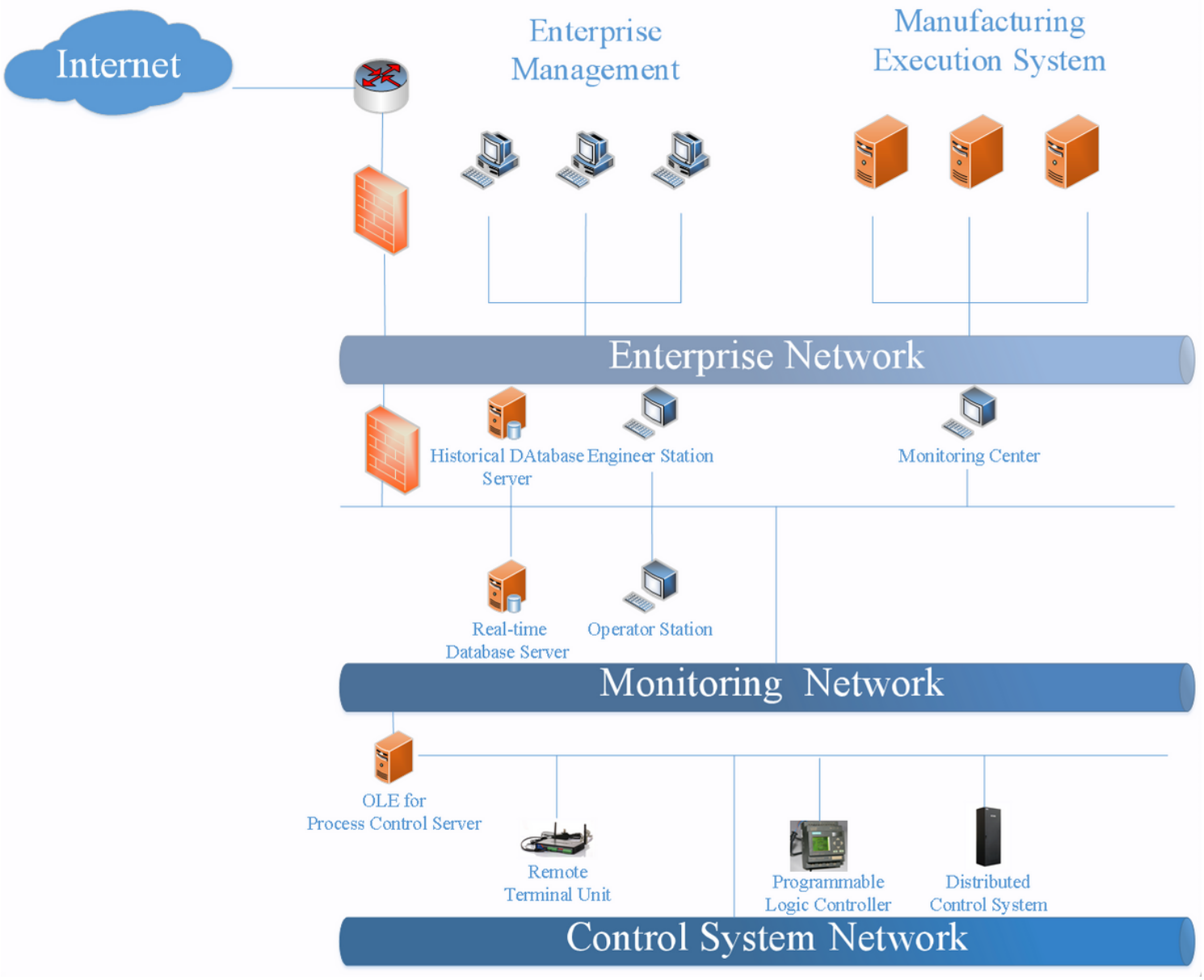

Figure 1

A Typical Industrial Internet Control System Architecture

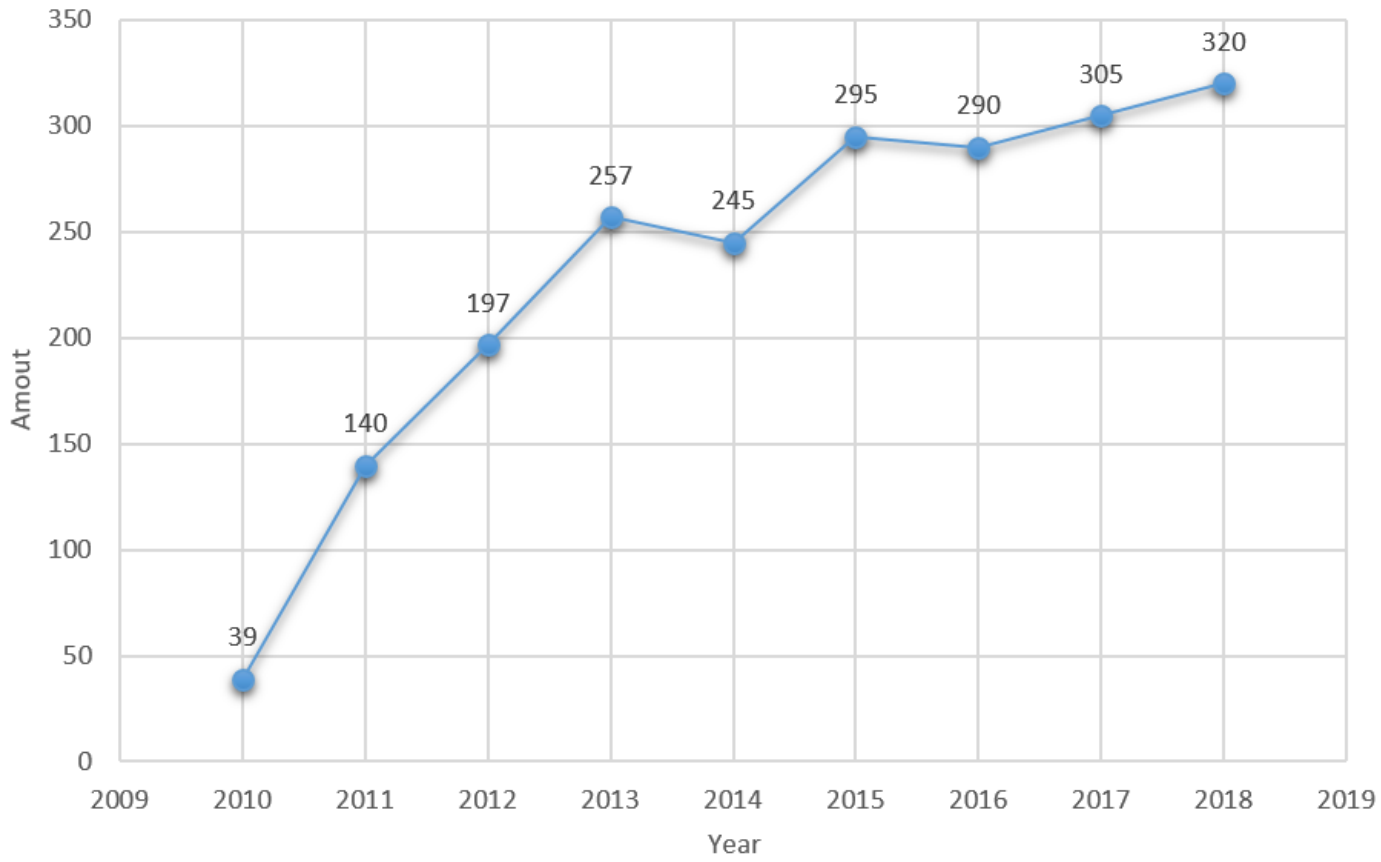

Figure 2

Historical Security Incidents of IIS

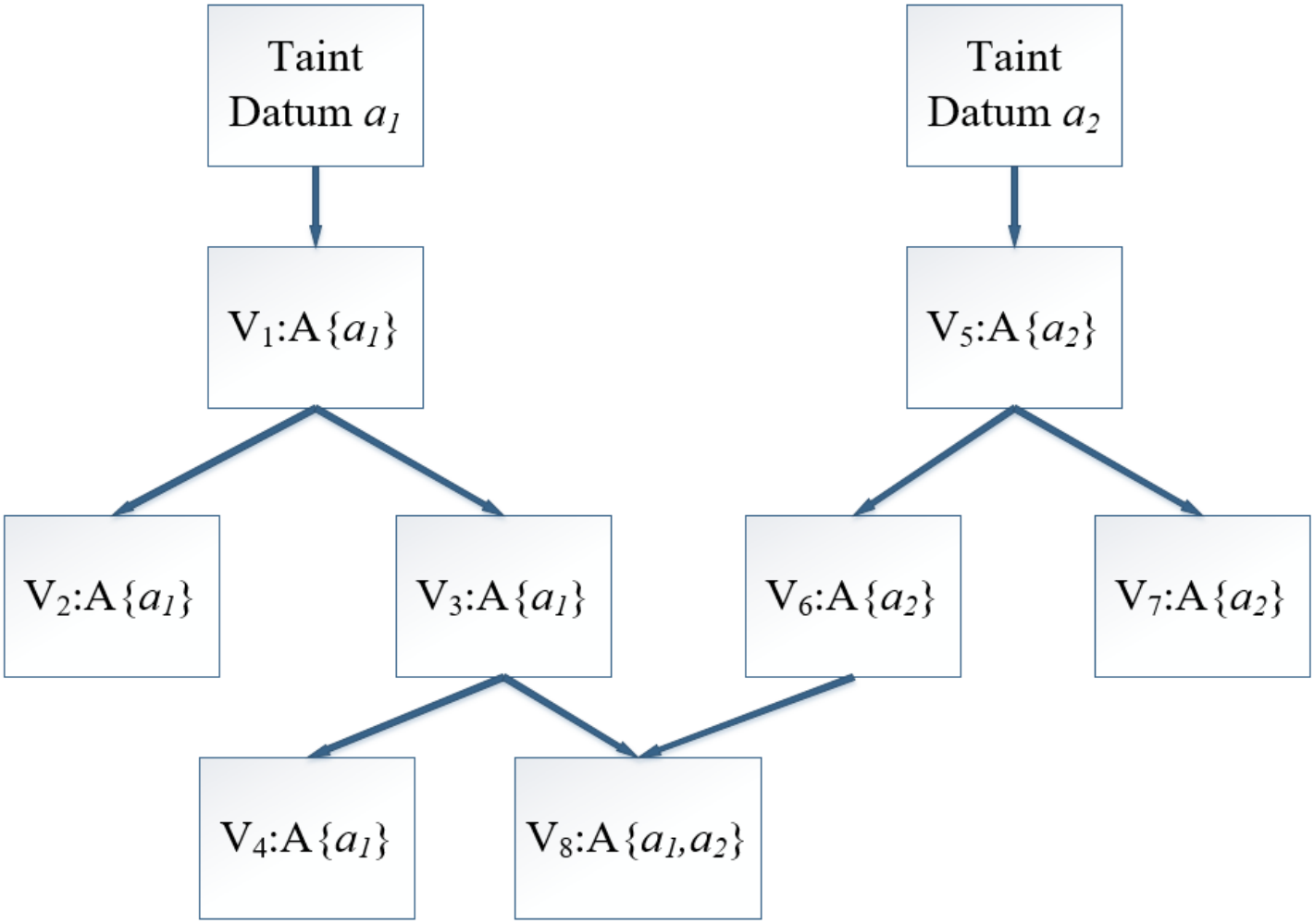

Figure 3

An Example of Dynamic Taint Analysis

```
1  begin
2  L1 x = get_input();
3  L2 msg = uri =' ';
4  L2 if (x == 'a'){
5  L4 uri = 'post';
6  L5 msg = 'a';
7  L6}
8  L7 else if (x == 'b'){
9  L8 uri = 'post';
10 L9 msg = 'b';
11 L10}
12 L11 send(uri, msg);
13 end
```

Figure 4

Example Code for Dynamic Analysis

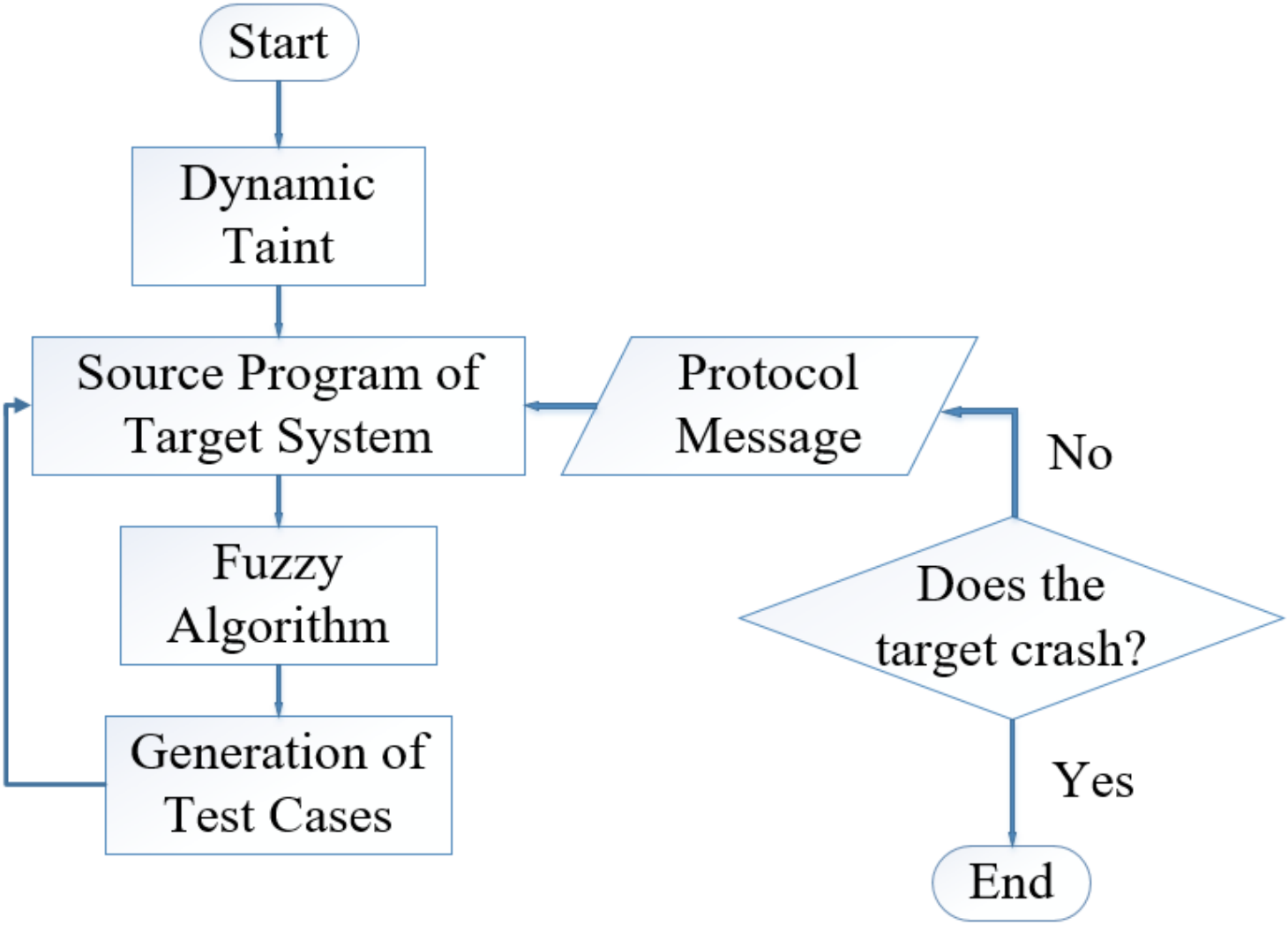

Figure 5

Industrial Internet Protocol Fuzz test Process Based on Dynamic Analysis

Identity (2)	Length (2)	Function Code (1)	Start Address (n)	End Address (n)	Data (n)	CRC (2)

Figure 6

An Example of Industrial Internet Protocol Input Format

```
int modbus_handle(char* field)
{
    if(field[0]!='\x00' || field[1]!='\x00') {
        return 0;
    }
    else {
        unsigned short length=get_length(field);
        if(get_crc(field)!=crc16(field,length)) {
            return 0;
        }
        else {
            char func=field[4];
            if(func=='\x01'){...}
            else if(func=='\x05'){
                unsigneds short s=get_start(field,2);
                unsigneds short e=get_end(field,2);
                if(s>e) {
                    return 0;
                }
                else {
                    for(int i=0;i<=e-s;i++) {
                        if(length-7<(e-s+1)*DATA_SIZE)
                            error_trigger();
                        if(s<=TRIG_POINT && TRIG_POINT<=e)
                            if(buf[9+TRIG_POINT-s]==KEY))
                                error_trigger();
                    write_register(s+i,field[9+i]);
                        }
```

Figure 7

An Example of Industrial Internet Protocol Code Realization

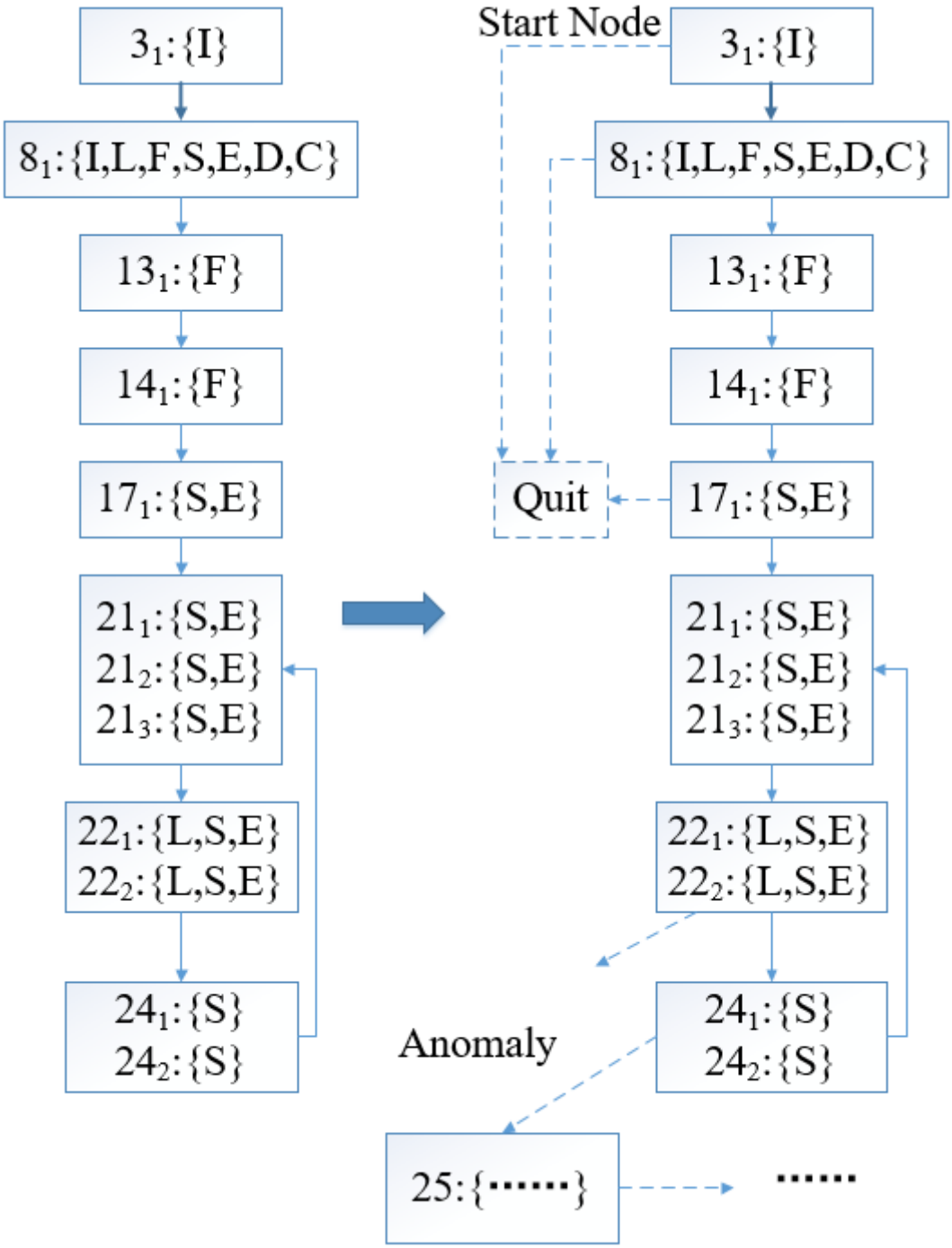

Figure 8

Control Flow Diagram and Execution Paths of the First Input

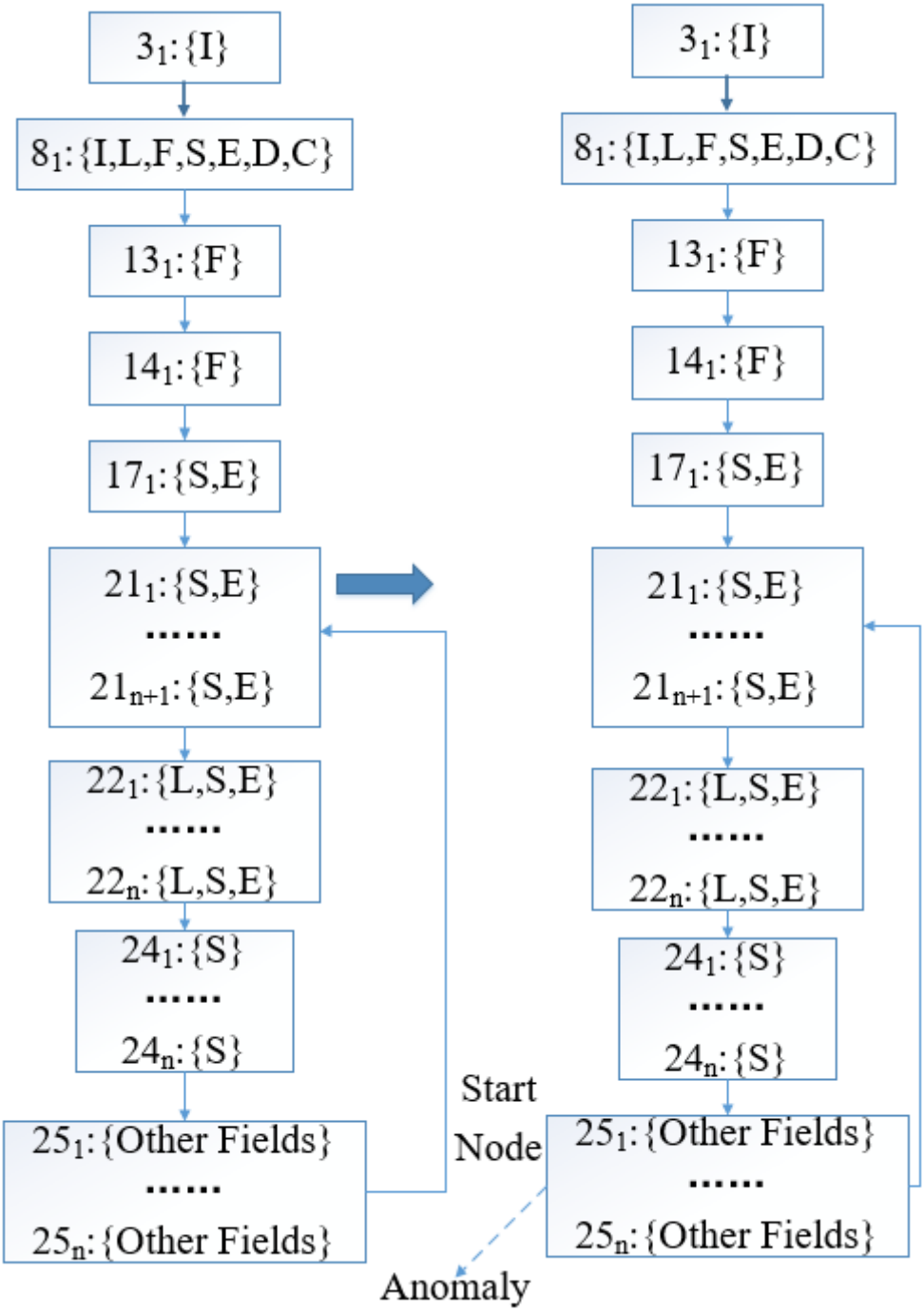

Figure 9

Control Flow Diagram and Execution Paths of the Second Input

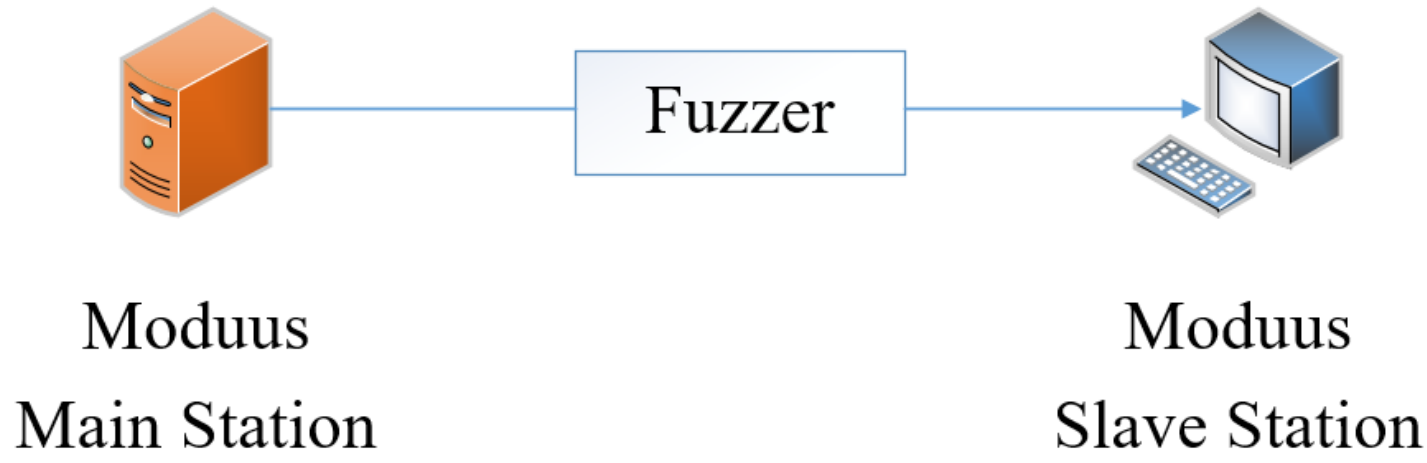

Figure 10

Test Environment

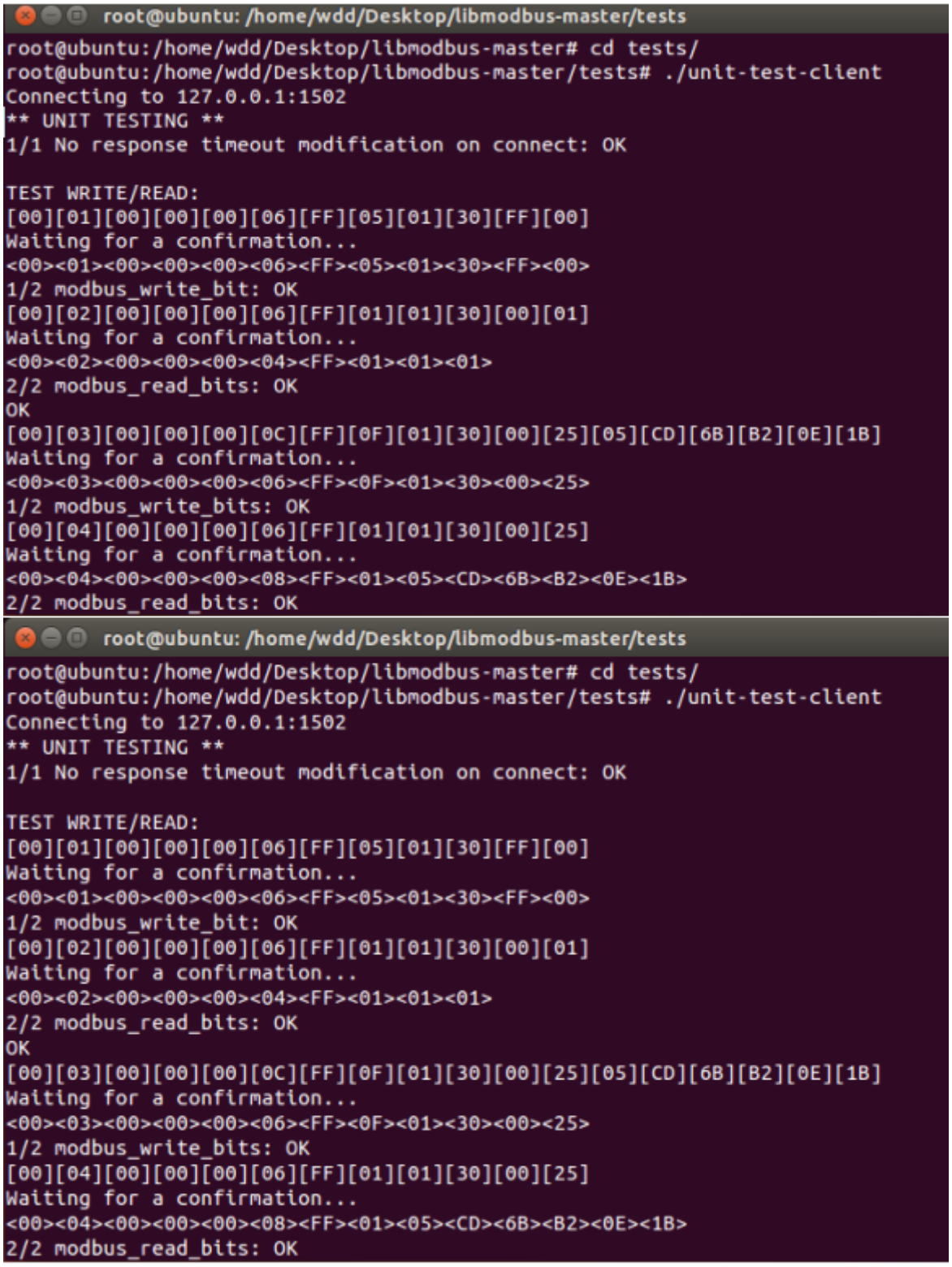

Figure 11

Test Environment Deployment

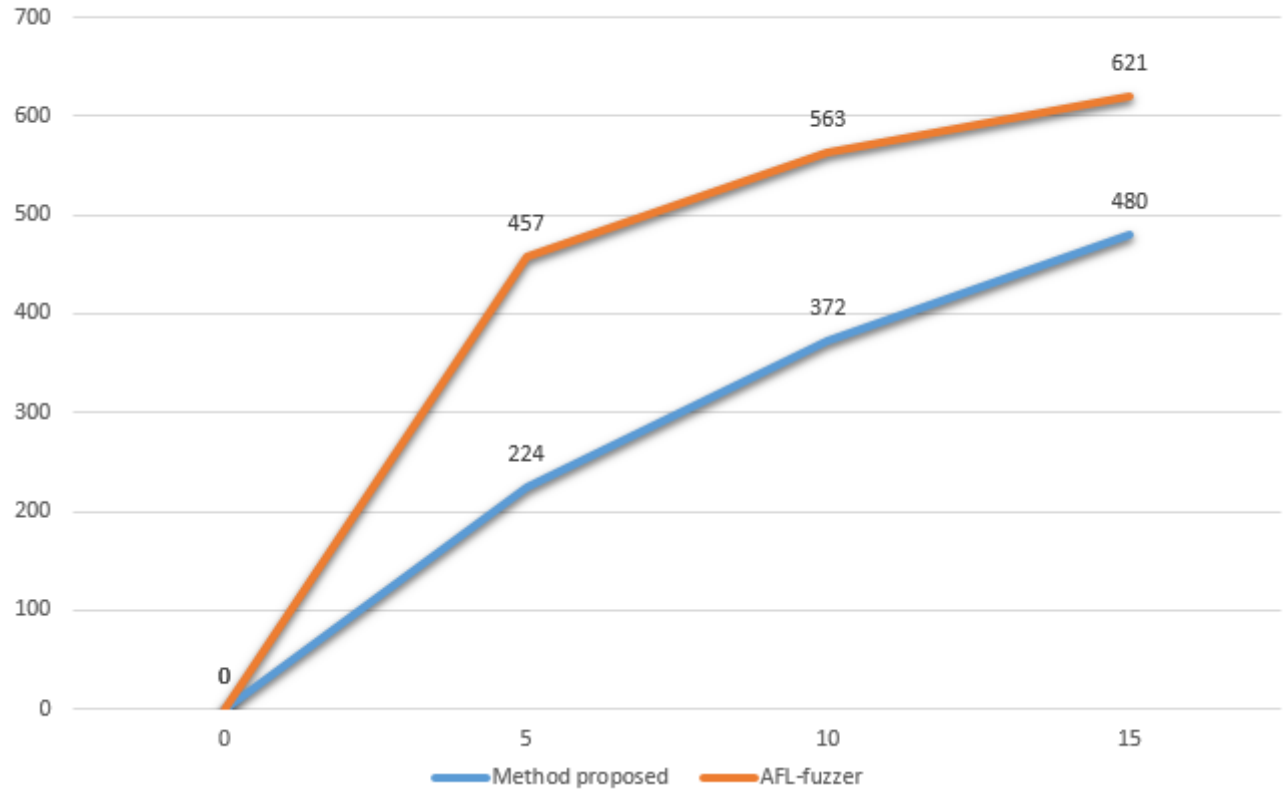

Figure 12

The Numbers of Test Cases with Different Numbers of Samples

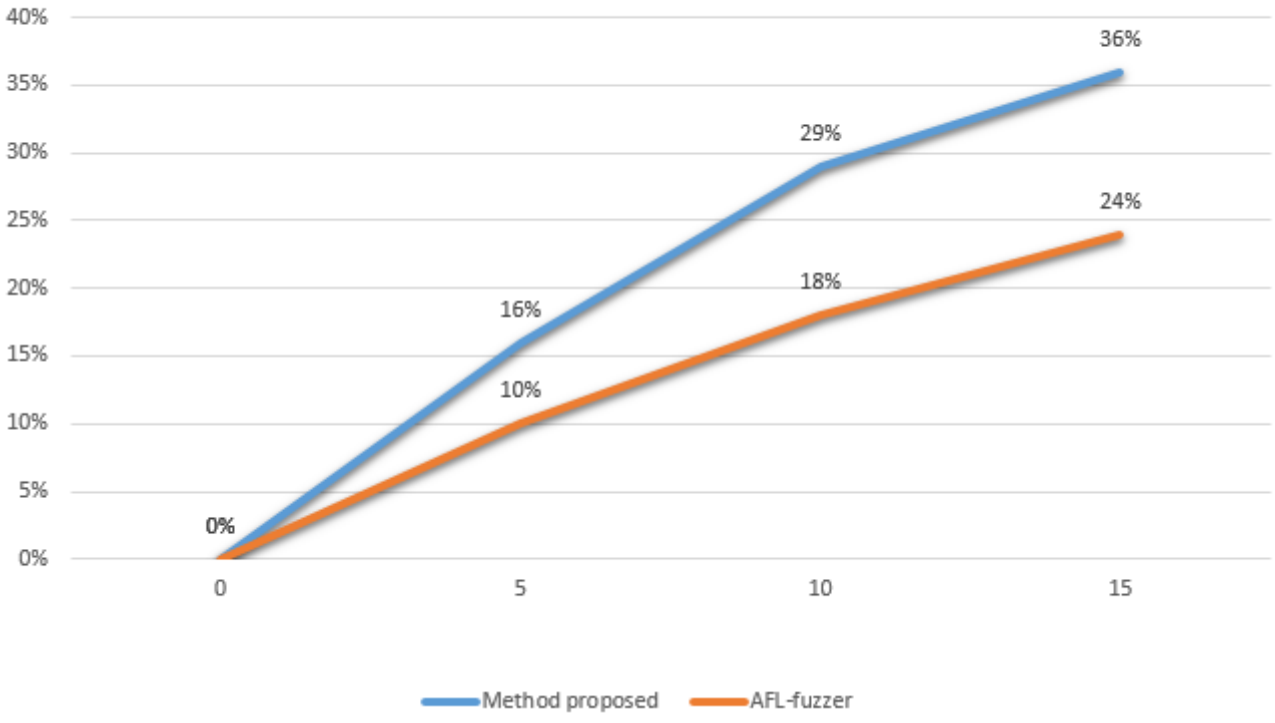

Figure 13

Code Coverage Rates with Different Numbers of Samples