A New Fuzzing Method Using Multi Data Samples Combination

*Xueyong Zhu [1] Zhiyong Wu [2]
[1,2]Network Information Center, University of Science and Technology of China, Hefei, Anhui, China
zhuxy@ustc.edu.cn, wuzhiyong0127@gmail.com

J. William Atwood[3]
[3]Department of Computer Science and Software Engineering, Concordia University, Montreal, Quebec, Canada
bill@cse.concordia.ca

Abstract-- Knowledge-based Fuzzing technologies have been applied successfully in software vulnerability mining, however, its current methods mainly focus on Fuzzing target software using a single data sample with one or multi-dimension input mutation [1], and thus the vulnerability mining results are not stable, false negatives of vulnerability are high and the selection of data sample depends on human analysis. To solve these problems, this paper proposes a model named Fuzzing Test Suite Generation model using multi data sample combination (FTSGc), which can automatically select multi data samples combination from a large scale data sample set to fuzz target software and generate the test cases that can cover more codes of the software vulnerabilities. To solve Data Sample Coverage Problem (DSCP) in the proposed FTSGc, a method of covering maximum nodes' semantic attributes with minimum running cost is put forward and a theorem named Maximum Coverage Theorem is given to select the data sample combination. We conclude that DSCP is actually the Set Covering Problem (SCP). Practical experimental results show that the proposed Fuzzing method works much better than the other current Fuzzing method on the Ability of Vulnerability Mining (AVM).

Key words: Fuzzing; Vulnerability mining; FTSGc; DSCP; AVM

I. INTRODUCTION

Fuzzing technology is a kind of dynamic vulnerability mining technology, and its key problem is to generate high semi-valid [2] test cases which can pass checks & verifications in programs and can cover more cases in the codes. Such checks & verifications often exist in practical software, for example, fixed fields, checksum, length counting, number counting, encoding and decoding, hash computation and encryption & decryption.

References [3], [4] and [5] combine static analysis technique, symbolic execution technique and concolic testing technique with Fuzzing technology, and they start fuzzing after achieving a better code coverage. Because many checks & verifications are mathematically designed and it is very hard to pass them, so these techniques cannot effectively pass these checks & verifications.

References [6] and [7] try to use genetic algorithms to improve code coverage and then start Fuzzing on target software, while genetic algorithm is only used as an advanced intelligent random search algorithm, so it is also hard to pass these strong checks & verifications and the Fuzzing technology with genetic algorithm is only applicable to some simple experimental programs but useless to the practical software programs.

Fuzzing tool FileFuzz [12] improves the number of semi-valid test cases discovered [2] by mutating and generating test cases based on a correct data sample, but because it considers nothing about the types, semantic attributes and the constraints among input elements, the test-space is large and the number of semi-valid test cases is still very low, and thus a lot of invalid test cases are generated which affects the Fuzzing effectiveness.

Fuzzing tool Autodef [10] mutates and generates test cases based on the automatic analysis on data samples. Although this method mutates data sample based on analyses, which improves the semi-validity of test cases and reduces value space to test, because current protocol automatic analyzing technique is not mature yet, the analyzed field location and size are not accurate and the constraints among input elements may be wrong, the semi-validity of generated test cases is still not good.

Reference [1] and the Fuzzing tools such as Peach[8], Sulley[9] analyze input elements in the data sample and the relationships among input elements according to achieved accurate file format or network protocol knowledge, and start Fuzzing software based on these analyzed results. These knowledge-based Fuzzing technologies can generate the test cases of high semi-validity; however, they only consider Fuzzing based on a single data sample and have nothing to do with how to select automatically the data sample combination from a large scale data sample set. Normally, a single data sample only covers parts of file format or network protocol, so it is impossible for them to cover the codes that execute other file format or protocol knowledge and it

* Corresponding Author

doi:10.4304/jcp.6.5.881-888

is impossible to mine the vulnerabilities in these codes. These technologies only can generate test cases that are easy to pass checks & verifications in software, but still cannot thoroughly achieve the high Fuzzing code coverage. There are some inborn defeats in these Fuzzing technologies, such as the effect of mining vulnerability is not stable and false negatives of vulnerability are high. To solve these problems and defeats, based on **FTSG** model in [1], a new knowledge-based Fuzzing test suite construction model (**FTSGc**) is proposed, which can automatically select a data sample combination from large scale data sample set and implement Fuzzing on target software.

II. FTSGC MODEL

The formal description of **FTSGc** could be described as follows:

$$\begin{aligned}&\textbf{FTSGc}=<\boldsymbol{S},\boldsymbol{A},\boldsymbol{C},\textbf{OP},\textbf{Result}>,\\&\boldsymbol{S}=\{s_1,s_2,\ldots,s_k,\ldots,s_n\},\\&\textbf{OP}=\{Tr_1,Tr_2,\boldsymbol{M},Slv\},\\&\textbf{Result}=\{S',sampletree_k,\boldsymbol{N_k},mediumtree,\\&\qquad newtree,testcase,testsuite\}.\end{aligned}\tag{1}$$

In which:

s_k, kth data sample of input elements to a target software mined, $\boldsymbol{S}$ is a primitive set of the data sample.

$\boldsymbol{A}$, a set of semantic attributes, $\boldsymbol{A}=\{a_1,a_2,\ldots,a_i,\ldots,a_p\}$, a_i is the semantic attribute of a node in a data sample tree structure. Semantic attribute describes the value type, space, and constraints of the node. Semantic attributes are independent of each other.

$\boldsymbol{C}$, a set of constraint condition, represents the constraints of semantic attributes, such as checksum and so on.

$sampletree_k$, a tree of a serial running data sample in the target software, is executed from s_k according to file format or protocol knowledge, the relationship between $sampletree_k$ and s_k is one to one, all the nodes in $sampletree_k$ constitute the set $\boldsymbol{N_k}$.

Tr_1 and Tr_2 are two different transformers form s_k to $sampletree_k$,, $\boldsymbol{M}$ is a set of mutators, Slv is a constraint solver. **OP** is a set of above operators. The strategy that **FTSGc** generates test cases by Tr_1, Tr_2, $\boldsymbol{M}$ and Slv shows in Fig. 1.

$S' \subseteq \boldsymbol{S}$, S' is a set constituted by representative elements selected from $\boldsymbol{S}$, and S' is the data sample combination to be used to generate test cases.

Tr_1 transforms s_k into $sampletree_k$, m_i mutates the nodes in $sampletree_k$ and generates $mediumtree$, Slv modifies corresponding nodes' values to satisfy all the constraints in $\boldsymbol{C}$ and then generates a $newtree$. The leaf nodes of the $newtree$ constitute a $testcase$, all the $testcases$ constitute a set $testsuite$.

Let $|m_i(sampletree_k)|$ be used to express the number of test cases generated by mutation operator m_i that mutates $sampletree_k$, and the total number of the test cases is:

```
1.  M = {m1, ..., mi, ..., mw }
2.  testsuite = {}
3.  for (each sk in S')
4.  {
5.     sampletreek = Tr1 (sk)
6.     for (each mi in M except GAMutator)
7.     {
8.        MTS = { mediumtree1, mediumtree2, …,
       mediumtreei,…} = mi(sampletreek)
9.        for (each mediumtreei in MTS)
10.          {
11.          newtreei=Slv(mediumtreei, C)
12.          testcasei= Tr2 (newtreei)
13.        Add testcasei into testsuite
14.          }
15.    }
16. }
17. run every element in testsuite in the target
    software and monitor them
```

Fig. 1 The strategy that FTSGc generates test cases

$$D = \left|testsuite\right| = \sum_{s_k \in S'} \sum_{i=1}^{w} \left|m_i(Tr(s_k))\right| \tag{2}$$

Every $sampletree_k$ is homologized a node set N_k, $N_k=\{n_1,n_2,\ldots,n_j,\ \ldots,n_{k'}\}$, every node has one semantic attribute, and the relationship between semantic attributes and node is multi-to-one. The semantic attribute of n_j is a_i and $a_i \in \boldsymbol{A}$. Mutation operator m_i travels all the nodes in N_k and operates every node, and will return a $mediumtree$ while it changes the value of a node each time. The total number of test cases that m_i operates $sampletree_k$ is the same as m_i operates N_k. Then:

$$|m_i(sampletree_k)|=|m_i(N_k)|=\sum_{j=1}^{k'}\left|m_i(n_j)\right| \tag{3}$$

and

$$D = \left|testsuite\right| = \sum_{s_k \in S'} \sum_{i=1}^{w} \sum_{j=1}^{k'} \left|m_i(n_j)\right| \tag{4}$$

In addition, m_i will implement mutation operation on n_j according to its semantic attribute a_j.

Line 11 in Fig. 1 shows that **FTSGc** modifies related inputs' values by Slv after mutating test cases based on data sample to satisfy the constraints, i.e. the members of $\boldsymbol{C}$. $\boldsymbol{C}$ can be generated according to the amount of

achieved knowledge. Because each mutation operation just mutates one node based on correct data sample, even if achieved knowledge is not enough, the generated test cases still can keep a certain semi-valid [2]. Line 3 in Fig. 1 shows that **FTSGc** constructs test cases based on multi data samples. **FTSGc** uses a tree structure to express data sample, because the tree structure can correctly express the sequential or nested relationships among elements in network protocols or file format. By mutating leaf nodes, **FTSGc** can test the codes that interpret and execute nodes' semantic attributes; by mutating non-leaf nodes, **FTSGc** tests the codes that deal with file format or protocol knowledge and thus improves code coverage. **FTSGc** mutates data sample with the consideration of constraints among input elements and the generated test cases can pass strong validations in program.

The advantages of **FTSGc** model includes: (1) the semi-validity of constructed test cases is high, the automation of Fuzzing process is high, and it could implement flexible and different depth Fuzzing on target software according to achieved knowledge; (2) This model generates test cases based on multi data sample combination, and it can effectively resolve some problems, such as, it is hard to select data sample combination automatically, Fuzzing effects depend too much on selected data samples, Fuzzing results are not stable and it is easy to miss partial vulnerabilities.

The biggest difference between **FTSGc** and other Fuzzing test cases generation tools (like Peach [8], Sulley [9]) using single data sample is that **FTSGc** generates test cases use multi data sample combination. The new problem it proposed is how to select a representative data sample combination from the primitive data sample set $\boldsymbol{S}$ to form a new set S', based on which **FTSGc** could achieve a good Fuzzing effect with minimum running cost and max code coverage. This new problem is called the data sample coverage problem (DSCP).

III. DSCP Problem

3.1 Analyses and Hypotheses of DSCP

The target software dealing with file format and network protocol must meet the requirements of document or protocol specification and definition, for example, the software of processing png graph must meet the specifications in [17] and [18]. Target software will execute every node according to every node's semantic attribute and initial values.

Hypothesis 1: There are surely some codes that they are especially used to execute a semantic attribute a_i, (i = 1,2,…,p) in target software.

$\boldsymbol{A}$ in **FTSGc** can be derived from two aspects: one is specifications which the target software has to follow, such as RFC documents; and the other one is by analyzing normal inputs (like network data package) based on reverse engineering. If there are not codes dealing with node whose semantic attribute is a_i, either the target software does not meet the corresponding specification or the analyzed inputs are not normal inputs. In fact, the two situations are all impossible, and there must be some codes in target software to execute some semantic attributes. On the other hand, if there are not such codes parsing some semantic attribute, then this semantic attribute is not necessary or the parsing function of target software is not full. So there are surely some codes in target software parsing the semantic attributes in $\boldsymbol{A}$.

Hypothesis 2: The tested target codes while the m_i operates n_j only relate to the nodes' semantic attributes but other information such as initial values of the nodes and so on.

In practical Fuzzing process, the codes **FTSGc** tests have a few relationships with nodes' initial values and the structure of the sub-tree whose root is this node, and relate mainly to the node's semantic attribute. Based on the node with the same semantic attribute from various data samples, the codes tested by **FTSGc** may have some little difference, which it can be ignored here. This is a reasonable hypothesis.

3.2 Definitions and Symbols of DSCP

The definitions of some symbols are as follows:

t_k, the set of all the semantic attributes of the nodes in the kth $sampletree_k$, and

$$t_k = \{ a_{k,1}, a_{k,2}, \ldots, a_{k,k''} \} \quad (5)$$

s_k, $sampletree_k$ and t_k are one to one map.

ATS, the set of t_k corresponding to every data sample,

$$\mathbf{ATS} = \{t_1, t_2, \ldots, t_k, \ldots, t_n\} \quad (6)$$

Obviously, $t_k \in \mathbf{ATS}$ (7)

$$t_k \subseteq \boldsymbol{A} \quad (8)$$

$number_k$, the vector of the numbers of the nodes, $n_{k,1}$ is the number of the nodes whose semantic attribute is $a_{k,1}$, so,

$$number_k = <n_{k,1}, n_{k,2}, \ldots, n_{k,k''}> \quad (9)$$

$\bigcup_{s_k \in S} t_k$, the set of semantic attributes covered by $\boldsymbol{S}$, and that is

$$\bigcup_{s_k \in S} t_k \subseteq \boldsymbol{A} \quad (10)$$

Let $$B = \left| \bigcup_{s_k \in S} t_k \right| \le p \quad (11)$$

We have given S', some sub-set of $\boldsymbol{S}$,

$$S' \subseteq \boldsymbol{S} \quad (12)$$

The set of semantic attributes covered by S' is $\bigcup_{s_k \in S'} t_k$,

$$\bigcup_{s_k \in S'} t_k \subseteq \bigcup_{s_k \in S} t_k \subseteq \boldsymbol{A} \quad (13)$$

c_i, the set of the codes in target software to execute the node whose semantic attribute is a_i, and there is a map relationship between a_i and c_i

$$c_i = \mathrm{f}(a_i) \ (i=1,2\ldots,p) \quad (14)$$

C_k, the set of codes tested by all the test cases are generated by **FTSGc** based on s_k

$price_k$, the vector of running prices of all the nodes′

semantic attributes from the kth $sampletree_k$, and

$$price_k = <p_{k,1}, p_{k,2}, \ldots, p_{k,k''}> \quad (15)$$

Let **P**(x) be the mapping function between x and the running price of x, then:

$$\mathbf{P}(a_{k,i}) = p_{k,i} \quad (16)$$

$$\mathbf{P}(t_k) = \sum_{1 \le i \le k''} p_{k,i} \quad (17)$$

ct, (complete testing), is a process of all the mutation operators m_i mutating the nodes in $sampletree_k$ and testing c_i; according to Fig. 1, every node will be implemented a *ct*.

The first method to compute $p_{k,i}$: compute the running price of $a_{k,i}$ with the total number of *ct* that implemented on all nodes, and

$$\mathbf{P}(a_{k,i}) = p_{k,i} = n_{k,i} \quad (18)$$

Explanation: from the operation process of m_i to $sampletree_k$, **FTSGc** will implement one and only one *ct* on every node, so the number of nodes whose semantic attribute is $a_{k,i}$ is the number of the *ct* on f($a_{k,i}$), and formula (18) is rational.

The second method to compute $p_{k,i}$: compute the running price of $a_{k,i}$ with the number of test cases generated by all the mutation operators implementing all the nodes whose semantic attribute is $a_{k,i}$,

$$\mathbf{P}(a_{k,i}) = \sum_{n_i \in \mathbf{N}_k, n_i.a = a_{k,j}.} \sum_{1 \le j \le w} |m_j(n_i)| \quad (19)$$

Explanation: The second way to compute running price of semantic attribute is with smaller granularity, because the test cases' numbers of different nodes with the different semantic attributes maybe different from each other largely.

Sometimes it is hard to compute the number of test cases generated by some mutation operators before they run, such as, the intelligent mutation operator *GAMutator* in reference [1], and then we should use the first method, otherwise the second method is much fitter.

3.3 Mathematical Model of DSCP

Lemma 1: if $a_i \in \boldsymbol{A}$, $a_j \in \boldsymbol{A}$, $i = 1,2,\ldots,p$, $j = 1,2,\ldots,p$, $i \neq j$, then

$$c_i - (\bigcup_{a_j \in (\boldsymbol{A} - a_i)} c_j) \neq \phi \quad (20)$$

Proof: According to hypothesis 1, there must be a segment of codes belong to c_i and used to execute a_i, and since semantic attributes are linear independent of each other, so Lemma 1 exists.

Lemma 2: $$C_k = \bigcup_{a_i \in t_k} c_i = \bigcup_{a_i \in t_k} \mathrm{f}(a_i) \quad (21)$$

$$\bigcup_{s_k \in S'} C_k = \bigcup_{s_k \in S'} (\bigcup_{a_i \in t_k} c_i) \quad (22)$$

According to hypothesis 2, it is easy to get Lemma 2.

Lemma 3: if $$\bigcup_{s_k \in S'} t_k \subset \bigcup_{s_k \in S} t_k \ , \quad (23)$$

then $$\bigcup_{s_k \in S'} C_k \subset \bigcup_{s_k \in S} C_k \ ; \quad (24)$$

Proof: to get any $$c_i \subseteq \bigcup_{s_k \in S'} C_k \ , \quad (25)$$

there is $$a_i \in \bigcup_{s_k \in S'} t_k \ , \text{ where } c_i = \mathrm{f}(a_i), \quad (26)$$

because: $$\bigcup_{s_k \in S'} t_k \subset \bigcup_{s_k \in S} t_k \ , \quad (27)$$

so $$a_i \in \bigcup_{s_k \in S} t_k \ , \quad (28)$$

according to (10), there is:

$$a_i \in \bigcup_{s_k \in S} t_k \subseteq \boldsymbol{A}, \quad (29)$$

According to (22):

$$c_i \subseteq \bigcup_{s_k \in S} C_k \quad (30)$$

According to (25), (30), there is

$$\bigcup_{s_k \in S'} C_k \subseteq \bigcup_{s_k \in S} C_k \quad (31)$$

Because(29), (23), to get any

$$a_i \in \bigcup_{s_k \in S} t_k - \bigcup_{s_k \in S'} t_k \ , \text{ in which } c_i = \mathrm{f}(a_i) \quad (32)$$

According to Lemma 1, there is:

$$c_i - (\bigcup_{a_j \in (\boldsymbol{A} - a_i)} c_j) \neq \phi \quad (33)$$

According to (13) and (32), there is:

$$\bigcup_{s_k \in S'} t_k \subseteq \boldsymbol{A} - a_i \quad (34)$$

so $$c_i - (\bigcup_{s_k \in S'} (\bigcup_{a_j \in t_k} c_j)) \neq \phi : \quad (35)$$

according to (22) and (35), there is:

$$c_i - (\bigcup_{s_k \in S'} C_k) \neq \phi \quad (36)$$

By (32), obviously $$c_i \subset \bigcup_{s_k \in S} C_k \quad (37)$$

Then, $$\bigcup_{s_k \in S} C_k - \bigcup_{s_k \in S'} C_k \neq \phi \quad (38)$$

so $$\bigcup_{s_k \in S} C_k \neq \bigcup_{s_k \in S'} C_k \quad (39)$$

According to(31) and (39), Lemma 3 is proved.

Lemma 4: if $$\bigcup_{s_k \in S'} t_k = \bigcup_{s_k \in S} t_k \ , \quad (40)$$

Then $$\bigcup_{s_k \in S'} C_k = \bigcup_{s_k \in S} C_k \quad (41)$$

Proof: to get any $$c_i \subseteq \bigcup_{s_k \in S'} C_k \ , \quad (42)$$

there is $$t \in \bigcup_{s_k \in S'} t_k \ , \text{ where } c_i = \mathrm{f}(a_i), \quad (43)$$

because: $\bigcup_{s_k \in S'} t_k = \bigcup_{s_k \in S} t_k$, (44)

so, $a_i \in \bigcup_{s_k \in S} t_k$, (45)

and according to (22):

$$c_i \subseteq \bigcup_{s_k \in S} C_k \quad (46)$$

$$\bigcup_{s_k \in S'} C_k \subseteq \bigcup_{s_k \in S} C_k \quad (47)$$

On the other hand,

$$\bigcup_{s_k \in S'} C_k \supseteq \bigcup_{s_k \in S} C_k \quad (48)$$

so: $\bigcup_{s_k \in S'} C_k = \bigcup_{s_k \in S} C_k$ 。 (49)

Maximum Coverage Theorem: $\forall S' \subseteq \boldsymbol{S}$, if and only if $\bigcup_{s_k \in S'} t_k = \bigcup_{s_k \in S} t_k$, then $\bigcup_{s_k \in S'} C_k$ gets the maximum value and it is $\bigcup_{s_k \in S} C_k$.

Proof: ① Sufficient proof: by Lemma 3 and Lemma 4, it is easy to prove. ②Necessity proof: using apagoge,

Assuming $\bigcup_{s_k \in S'} C_k = \bigcup_{s_k \in S} C_k$, (50)

But meanwhile $\bigcup_{s_k \in S'} t_k \neq \bigcup_{s_k \in S} t_k$ (51)

Then, it surely exists $\bigcup_{s_k \in S'} t_k \subset \bigcup_{s_k \in S} t_k$ (52)

According to Lemma 3:

$$\bigcup_{s_k \in S'} C_k \subset \bigcup_{s_k \in S} C_k \quad (53)$$

This is contradictory to assumption, so necessity proof is established. Combining ① and ②, Maximum Coverage Theorem is established.

According to Maximum Coverage Theorem, for any $S' \subseteq \boldsymbol{S}$, only while $\bigcup_{s_k \in S'} t_k = \bigcup_{s_k \in S} t_k$, $\bigcup_{s_k \in S'} C_k$ will get maximum value, and $\max(\bigcup_{s_k \in S'} C_k) = \bigcup_{s_k \in S} C_k$, at this time **FTSGc** will test most codes.

Obviously, based on S', testing along path in Fig. 1, the summarization of running price of **FTSGc** test is

$$P_{S'} = \sum_{s_k \in S'} \sum_{i=1}^{k''} p_{k,i} \quad (54)$$

So, the formal description of **DSCP** model is:

Inputs: formula (1), (5), (6), (15)

Output:

$$S' \quad (55)$$

Constraints:

$$\bigcup_{s_k \in S'} t_k = \bigcup_{s_k \in S} t_k \quad (56)$$

Object:

$$\text{Min} \quad \sum_{s_k \in S'} \sum_{i=1}^{k''} p_{k,i} \quad (57)$$

3.4 Solving of DSCP

Firstly we analyze the DSCP, and then solve it.

3.4.1 Analyses of DSCP model

Let

$$\bigcup_{s_k \in S} t_k = \{a_1, a_2, \ldots, a_i, \ldots, a_B\} \quad (58)$$

be the semantic attributes covered by $\boldsymbol{S}$.

Since the relationship between s_k and t_k is one to one, now just as well assuming $s_k = t_k$, so,

$$s_k \subseteq \{a_1, a_2, \ldots, a_i, \ldots, a_B\} \quad (59)$$

Combining with(17), then

$$\mathbf{P}(s_k) = \mathbf{P}(t_k) = \sum_{1 \le i \le k''} p_{k,i} \quad (60)$$

is the running price of s_k (for short $\mathbf{P}_k$), and

$$\mathbf{P}_k = \mathbf{P}(s_k) \quad (61)$$

Therefore, the model of **DSCP** could be transformed to solve the vector X = < x_1, x_2,…, x_k,…, x_n>. Here, let us introduce a medium variable $d_{i,k}$:

$$d_{i,k} = \begin{cases} 1, if\ a_i \in s_k, \\ 0, otherwise. \end{cases} \text{ where } i=1,\ldots,B\text{: } k=1,\ldots,n\text{: } (62)$$

$$x_k = \begin{cases} 1, if\ s_k \in S' \\ 0, otherwise \end{cases} \quad \text{where } k=1,\ldots,n\text{: } (63)$$

Constraints:

$$\sum_{1 \le k \le n} d_{i,k} x_k \ge 1, \quad k=1,\ldots,n\text{: } x_k \in \{0,1\}\text{: } (64)$$

Objective:

$$\min \sum_{k=1}^{n} \mathbf{P}_k * x_k \quad (65)$$

By above analyses, **DSCP** is actually a non-unicost set covering problem [13], which is a NP-complete problem [14].

3.4.2 Greedy algorithm to solve DSCP

Next, we use the greedy algorithm to solve the **DSCP** problem, and its description is as follow:

1. for (each s_k in $\boldsymbol{S}$)
   {
   Transform s_k to get t_k
   };
2. Let S' be null, LEFT = **ATS**, $UNCOV = \bigcup_{1 \le k \le n} t_k$;

3. If *UNCOV* is null, return *S'*;

4. For every $t_k \in$ LEFT, computing running price efficiency **PE** = $\frac{\sum_{1\le i\le k''} p_{k,i}}{|t_k \cap UNCOV|}$, get the t_j which minimizes **PE;**

5. Let $S' = S' \cup \{s_j\}$, $UNCOV = UNCOV - t_j$, LEFT = LEFT – { t_j };

6. go to step 3.

Using the method proposed in [15], combining with(11), it is easy to prove that the ratio of solution value between using the greedy algorithm and the optimum resolution is less than

$$H(B) = 1+\ln(B) \quad (66)$$

IV. EXPERIMENT

4.1 Method to evaluate Fuzzing effects

Code coverage can not directly reflect AVM (the Ability of Vulnerability Mining) of the Fuzzing [11], but there must be some vulnerabilities to be missed if Fuzzing can not reach a certain level code coverage. Reference [2] proposed the 'semi-valid' to describe the characteristics of test cases generated by Fuzzing technology, but the quantitative computing method have been not proposed. This paper proposed a quantitative definition of AVM to evaluate the effects of Fuzzing:

Definition: $AVM = w_1*c + w_2 * v^2$ (67)

w_1, w_2 is weight coefficients, where $w_1 + w_2 = 1$, $0 \le w_1, w_2 \le 1$; c is the code coverage; v is the number of vulnerabilities discovered in Fuzzing process.

4.2 Analyses of Practical Experiment

LibPng [16] is used to execute the pictures of png [17][18] format, which is applied widely in several operating systems like Unix, OS/2, Windows and Mac OS. The target software of **FTSGc** is LibPng.dll v1.0.6, which is compiled and generated by Visual Studio2008, and it is called by usePng.exe that we developed. Experiments platform: CPU, Petium(R) Dual-Core 2.50GHz; memory, 2.00GB; OS, Windows XP Service Pack 2; code coverage computation tool: Paimei[17] . According to paper [17] and [18], ***A***={PNG_ROOT, PNG_IHDR_CHUNK, …, PNG_IEND_CHUNK, PNG_SIGNATURE, width, height,…, iendCrc}, in which the element number is 177.

Primitive data sample set ***S*** in **FTSGc** is selected randomly from 10000 png[17][18] sample files, and experimenting data can be seen in following figures: (in these figures, G means greedy algorithm, R means random algorithm, U1 means (18), and U2 means (19).)

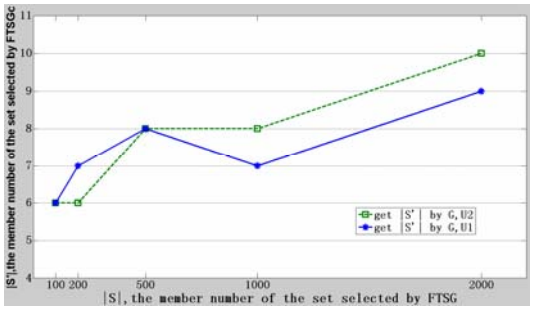

Fig. 2 selected data sample number |*S*'| from ***S***

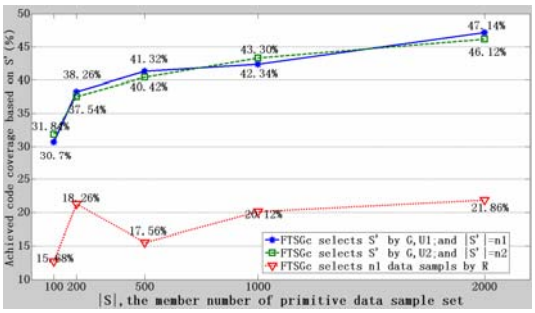

Fig. 3 code coverage achieved by FTSGc using different S.

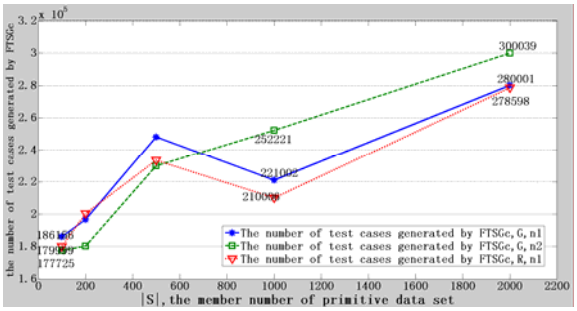

Fig. 4 the number of test cases FTSGc generated using different S

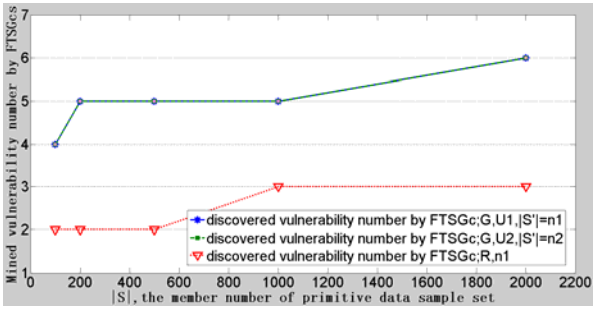

Fig. 5 the number of vulnerability mined out by FTSGc using different S

Fig. 2, Fig. 3, Fig. 4 and Fig. 5 illustrates the number of elements in *S'*, the code coverage on target software LibPng.dll, the number of which generated test case and the number of which mined vulnerability are increasing with the increase of the number of elements in ***S.*** With the same running cost (in Fig. 4), the code coverage (in Fig. 3, it used statement coverage [19][20] to calculate code coverage) and the number of mined vulnerabilities (in Fig.

5) based on greedy selection are higher than what based on random selection. On the other hand, Fig. 4 shows that the second measure method to calculate running price with (19) is better than the first method with (18); but the first method is more immediate and easy.

The AVM of **FTSG** using single data sample, **FTSGc** using greedy selection of multi data sample combination and using random selection of multi data sample combination individually show in TABLE 1. The column a1 in TABLE 1 is AVM of current Fuzzing method **FTSG**. The data of a1 are the best representative of using single data sample on all tested data samples, and based on this representative data sample **FTSG** have mined 2 vulnerabilities and achieved the code coverage of 15% with only 49992 test cases. The a2 is used to denote AVM of **FTSGc** using random selection of multi data sample combination. The a3 and a4 are individually used to denote AVM of **FTSGc** using greedy selection of multi data sample combination.

TABLE 1
the AVM of Fuzzing (w_1=w_2=0.5)

$\|S'\|$	Current Fuzzing Technology **FTSG**	**FTSGc** using random selection	**FTSGc** using greedy selection		Comparison	
	a1,using single data sample	a2,using S',(R)	a3,using S',(G, U1)	a4,usng S',(G, U2)	min(a3,a4)/a1	min(a3,a4)/a2
100	2.0750	2.0784	8.1535	8.1592	**3.9294**	3.9230
200	2.0750	2.0913	12.6913	12.6877	3.9459	3.9151
500	2.0750	2.0878	12.7066	12.7021	6.1215	6.0840
1000	2.0750	4.6006	12.7117	12.7165	6.1261	**2.7631**
2000	2.0750	4.6093	18.2357	18.2306	8.7858	3.9552

The data in TABLE 1 show that the ratio between the AVM of **FTSGc** using greedy selection multi data sample combination and the AVM of **FTSG** [1] using single data sample is bigger than 3.9294; In fact, based on **FTSGc**, various selection algorithms can be used to solve **DSCP,** such as random selection and greedy selection, which only results in different AVM. For example, the ratio between AVM of **FTSGc** using greedy selection multi data sample combination and the AVM of **FTSGc** using random selection multi data sample combination is bigger than 2.7631.

V. CONCLUSION

Knowledge-based Fuzzing technologies have been applied successfully in mining software vulnerability and are getting more and more attention; however, current Fuzzing methods are all using single data sample, and there are some problems such as depending too much on selected data sample, low code coverage and ease of missing some vulnerabilities. The proposed a Fuzzing test suite generation model named **FTSGc** using multi data sample combination has successfully solved these problems partly, and it is enable to select automatically data sample combination from a large data sample set. The proposed AVM can partly evaluate mining vulnerabilities effect. Its Fuzzing results on LibPng.dll v1.0.6 show that **FTSGc** method using greedy algorithm to select data sample combination performs much better than current Fuzzing techniques that do not consider multi data sample combination, and **FTSGc** is effective and practicable.

VI. ACKNOWLEDGEMENT

We would like to thank our research team members for their work and suggestions on this paper. We also acknowledge the support of the Natural Sciences Council of Anhui China, through its 090412055 program. J. William Atwood acknowledges the support of the Natural Sciences and Engineering Research Council of Canada, through its Discovery Grants program.

REFERENCES

[1] Zhiyong Wu, J. William Atwood, Xueyong Zhu. A New Fuzzing Technique for Software Vulnerability Mining, In Proceedings of the IEEE CONSEG'09. Chennai, India, Dec. 19, 2009.

[2] P. Oehlert. Violating Assumptions with Fuzzing [J]. IEEE Security & Privacy, Vol.3 (No.2), 2005, pages 58-62, In IEEE Computer Society.

[3] L. Andrea, M. Lorenzo, M. Mattia and P. Roberto. A Smart Fuzzer for x86 Executables[C]. In Proceedings of the Third International Workshop on Software Engineering for Secure Systems: IEEE Computer Society, 2007.

[4] P. Godefroid, M. Levin, and D. Molnar. Automated whitebox fuzz testing[C]. In NDSS, 2008.

[5] P. Godefroid, Peli de Halleux, Aditya V.,et al. Automating Software Testing Using Program Analysis [J], IEEE SOFTWARE, September/October 2008, pp. 30-37.

[6] LIU Guang-Hong, WU Gang, Zheng Tao, SHUAI Jian-Mei, TANG Zhuo-Chun. Vulnerability Analysis for X86 Executables Using Genetic Algorithm and Fuzzing[C]. In Third 2008 International Conference on Convergence Hybird Information Technology (ICCIT.2008.9).

[7] Sherri Sparks, Ryan Cunningham, Shawn Embleton, Cliff C. Zou. "Automated Vulnerability Analysis: Leveraging

Control Flow for Evolutionary Input Crafting", in 23rd Annual Computer Security Softwares Conference (ACSAC), p.477-486, Miami Beach, Florida, Dec. 10-14, 2007. (acceptance ratio: 40/191=21%).
[8] Peach[CP/OL].Webpage: http://www.peachFuzzer.com. http://peachfuzz.sourceforge.net.http://peachfuzz@googlegroups. com. Visited on June, 2009.
[9] Sulley[CP/OL]. Web page: http://www.fuzzing.org. Visited on June, 2009.
[10] AutoDafe[CP/OL].Webpage: http://autodafe.sourceforge.net/docs/autodafe.pdf. Visited on June, 2009.
[11] C. Miller, Z. N.J. Peterson. Analysis of Mutation and Generation-Based Fuzzing[EB/OL], March 1, 2007. Web page:http://securityevaluators.com/files/papers/analysisfuzzing .pdf, visited on June, 2009.
[12] FileFuzz[CP/OL] .http://labs.idefense.com/software/fuzzing.php Visited on Sept., 2009.
[13] Guanghui Lan, Gail W. DePuy, Gary E. Whitehouse. An effective and simple heuristic for the set covering problem. European Journal of Operation Research 176(2007). P.1387-1403.
[14] Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco, CA.
[15] David S.Johnson, Approximation algorithms for combinatorial problems. Proceedings of the fifth annual ACM symposium on theory of computing. P.38-49. April 30-May02, 1973, Austin, Texas, United States.
[16] LibPng[CP/OL]. http://www.libpng.org. Visited on Sept., 2009.
[17] T. Boutell, et al., PNG (Portable Network Grapics) Specification, Version 1.0[M/OL], IETF Request for Comments 2083.
[18] Greg Roelofs, PNG: The Definitive Guide[M/OL]. O′ REILLY. 2009.9. http://www.libpng.org/pub/png/book.
[19] Ntafos, Simeon,"A Comparison of Some Structural Testing Strategies", IEEE Trans. Software Eng., Vol.14, No.6, June 1988, pp.868-874.
[20] Beizer, Boris, "Software Testing Techniques", 2nd edition, New York: Van Nostrand Reinhold, 1990.

Xueyong Zhu, male, 52-year-olds, Professor. He works in Network Information Center of University of Science and Technology of China. His research is concerned with the Computer Network, Information Security, E-commerce, Communications Protocols and Database. He worked in High Speed Protocols Laboratory of Concordia University in Canada from Sept. 2004 to Oct. 2005.

Zhiyong Wu, male, 32-year-olds, Ph. D, lecturer, He works in Network Information Center of University of Science and Technology of China. His research is concerned with the Computer Network, Information Security, Software engineering, Communications Protocols. He received his B.Eng. (2002), his M.A.Sc. (2007), and his Ph.D. (2010) from the University of Science and Technology of China.

J. W. Atwood, male, 69-year-olds, Ph. D, Professor. IEEE member, He works in Department of Computer Science and Software Engineering Concordia University. His research is concerned with the specification, validation, and evaluation of communications protocols, software engineering, and with the application of these protocols within computer networks. He is Director of the High Speed Protocols Laboratory.