k-gram 기법을 이용한 프로그램 비교 방법

김영철

2016. 8. 3.

목차

1. 바이너리 코드 수준에서 연산자 k-gram 기반의 오픈소스 소프트웨어 탐지, 정보과학회논문지 2014. 2 최종천, 조성제
2. k-gram의 근사 매칭을 이용한 이진 프로그램의 비교 방법, 정보과학회논문지 2012. 4 임현일

바이너리 코드 수준에서 연산자 k-gram 기반의 오픈소스 소프트웨어 탐지

정보과학회논문지 2014. 2 최종천 조성제

서론

- 바이너리 기반 OSS 탐지 필요
- k-gram 방식을 적용한 탐지 기법 제안

관련 연구

- 바이트 코드 명령어를 이용한 k-gram 기법
- 소스코드 유사도 비교
- API 정보 기반 특징 정보를 추출한 버스마크 기법

제안 기법

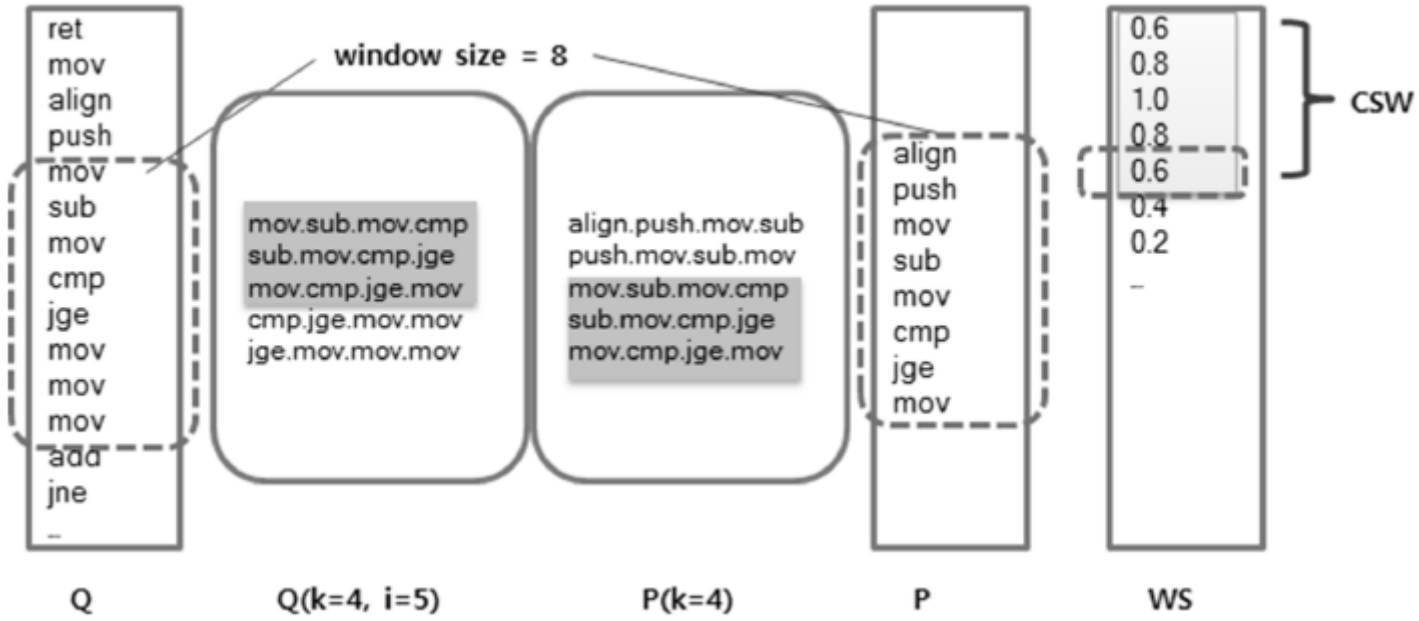
- 정적 라이브러리 기반 k-gram 추출
  - PE 파일은 .text 영역에 대하여 어셈블리 코드를 획득
  - 리눅스 시스템에서는 objdump를 이용하여 획득
  - 어셈블리 코드에서 연산자(opcode)만 추출하여 k-gram 기법 적용
  - OSS 탐지를 위해 정적 라이브러리 활용

제안 기법

- 바이너리코드기반 오픈소스 탐지기법
  - OSS 코드를 역어셈블하여 k-gram 버스마크 추출
  - 대상 프로그램에서 정한 구간 코드에 대한 연산자 추출
  - 대상 프로그램의 k-gram과 OSS 코드의 k-gram의 유사도 계산
  - 계산된 유사도에서 유효한 구간들을 탐지
  - 연속된 유사 구간이 여럿 존재할 경우, 최대 유사도를 반영하여 OSS가 포함된 구간으로 판정

제안 기법

- 바이너리코드기반 오픈소스 탐지기법

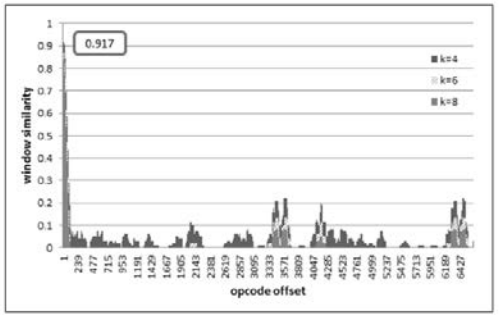


실험 및 평가

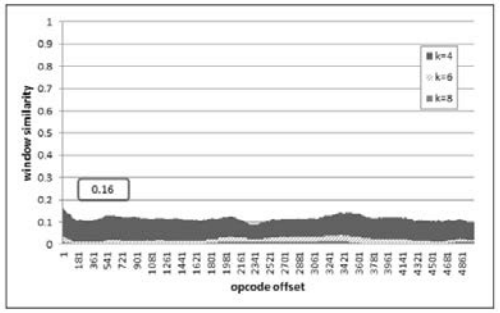
- 3개의 라이브러리와 라이브러리를 포함한 실행 파일로 실험

source name	binary type	# of source code lines	# of assembly code lines
qs_lib	lib	25 lines	95 lines
kdtree	lib	809 lines	1,733 lines
lzDec	lib	1,230 lines	2,591 lines
quicksort	exe	57 lines	6,746 lines
qs_test (qs_lib)	exe	35 + 25 lines	9,387 lines
kd_test (kdtree)	exe	88 + 809 lines	20,106 lines
7zDec (lzDec)	exe	501 + 1,230 lines	12,131 lines

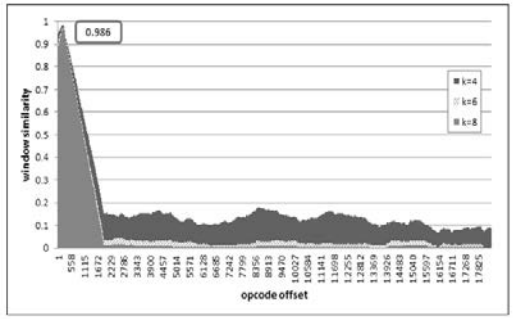
실험 및 평가



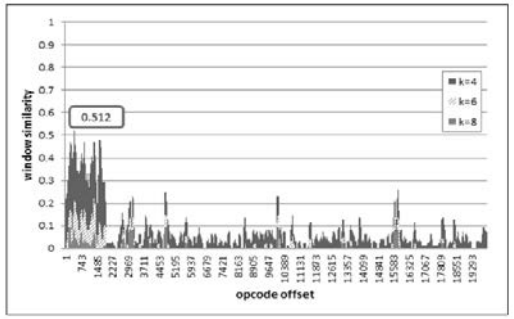
(a) quicksort vs. qs_lib



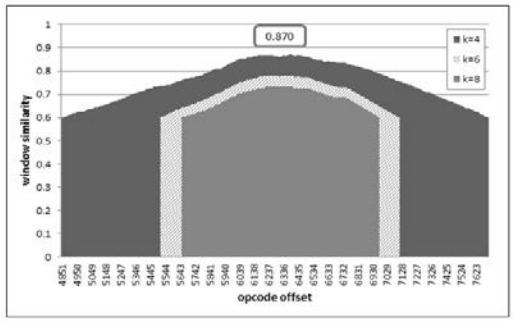
(b) quicksort vs. kdtree



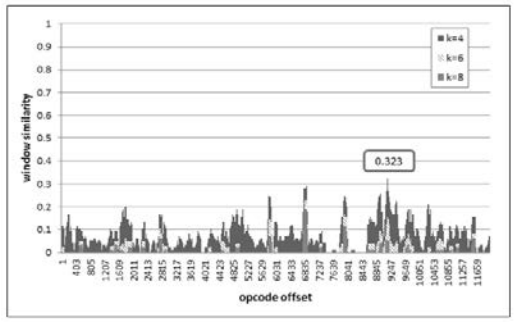
(a) kd_test vs. kdtree



(b) kd_test vs. qs_lib



(a) 7zDec vs. lzDec



(b) 7zDec vs. qs_lib

k-gram의 근사 매칭을 이용한 이진 프로그램의 비교 방법

정보과학회논문지 2012. 4 임현일

서론

- 소프트웨어의 유사성 분석
  - 소스 코드에 적용 가능한 방법
  - 이진 프로그램에 적용 가능한 방법
    - k-gram 기법 소개 + 신뢰성 향상을 위한 k-gram의 근사 매칭 방법 제안

관련 연구

- 표절 탐지를 위한 소스 코드 비교 방법
  - 소스 코드로부터 얻을 수 있는 토큰들의 구조, 시퀀스 비교
  - 정확성, 신뢰성이 높지만 소스 코드가 공개되어야 함.

- 이진 프로그램 비교 방법
  - k-gram 방법 – 명령어 재배치 등에 취약
  - k-gram의 동적 적용 – 실행 환경이 제공되어야 함.

k-gram 방법

- k-gram 방법을 통한 프로그램의 비교
  - 비교하기 위한 두 프로그램으로부터 k-gram을 추출
  - 동일한 k-gram이 얼마나 많은가를 평가

$$Similarity(P,Q) = \frac{|kgramS(P) \cap kgramS(Q)|}{\min(|kgramS(P)|, |kgramS(Q)|)}$$

k-gram 방법

- k-gram 방법의 향상 방안
  - 비교 대상의 변화가 없을 때 간단하고 효과적임
  - 작은 변화에 대한 민감도가 높음
  - 단점을 보완하기 위해 k-gram 근사 매칭을 적용
    - 수정이나 변화에 효과적으로 대응하는 유연성 확보
    - 유사 프로그램에 비교 결과에 대해 보다 안정적임

k-gram 근사 매칭을 이용한 소프트웨어의 유사성 분석

- k-gram의 근사 매칭
  - 일정 수준 유사성이 있는 k-gram에 대해서도 매칭에 반영
  - 두 k-gram의 비교 방법으로 LCS 방법 이용

```
match(kgram1, kgram2, threshold)
{
        len = length( LCS( kgram1, kgram2 ) );
        if ( len / k > threshold )
                return true;
        else
                return false;
}
```

k-gram 근사 매칭을 이용한 소프트웨어의 유사성 분석

- 프로그램의 유사성
  - 앞의 match함수를 이용하여 정확히 일치하는 k-gram 뿐만 아니라 일정 수준 이상 유사한 k-gram에 대해서도 매칭에 반영
  - k의 크기를 충분히 늘리는 것이 가능

```
compare(P, Q, threshold)
{
        count = 0;
        for kgram1 in kgramS(P)
                for kgram2 in kgramS(Q)
                        if ( match(kgram1, kgram2, threshold) ) {
                                count = count + 1;
                                break;
                        }
        Similarity = count / min(|kgramS(P)|, |kgramS(Q)|)
}
```

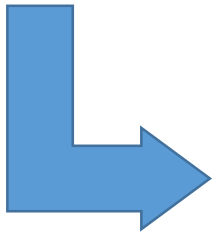
k-gram 근사 매칭을 이용한 소프트웨어의 유사성 분석

- k-gram의 근사 매칭을 통한 프로그램 비교

```
int fun1 (int i, int j){

        while(i < 10)
                i=i+ 1;
        return (i+ j);
}
```

```
int fun2 (int i, int j){
        if(i == j){
                return (i+ j);
        }
        else {
                i=i+ 1;
                return i;
        }
}
```



```
int fun1(int, int);
  Code:
    0:   iload_1
    1:   bipush   10
    3:   if_icmpge 13
    6:   iload_1          // i = i + 1;
    7:   iconst_1
    8:   iadd
    9:   istore_1
   10:   goto     0
   13:   iload_1          // return (i+j);
   14:   iload_2
   15:   iadd
   16:   ireturn
```

```
int fun2(int, int);
  Code:
    0:   iload_1
    1:   iload_2
    2:   if_icmpne 9
    5:   iload_1          // return (i+j);
    6:   iload_2
    7:   iadd
    8:   ireturn
    9:   iload_1          // i = i + 1;
   10:   iconst_1
   11:   iadd
   12:   istore_1
   13:   iload_1
   14:   ireturn
```

k-gram 근사 매칭을 이용한 소프트웨어의 유사성 분석

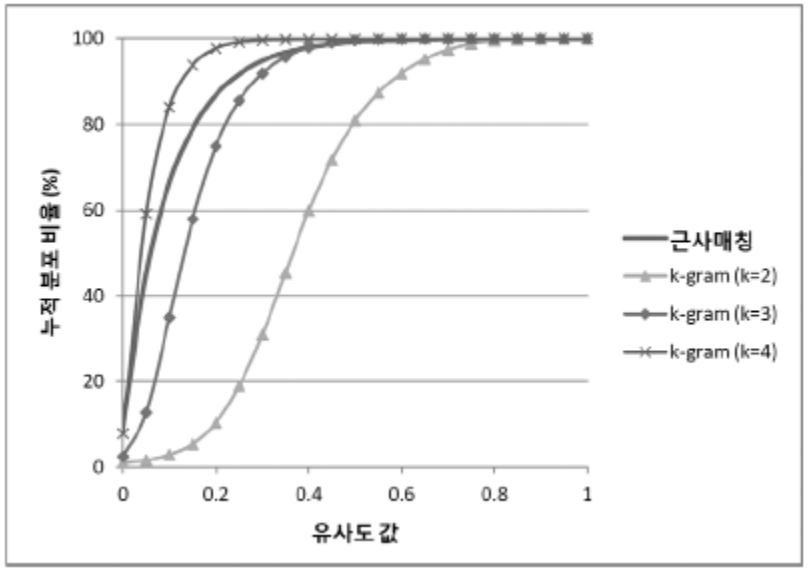
- 근사 매칭의 최적화
  - 적절한 임계값과 k의 크기를 적절하게 설정하는 것이 중요함

임계치	k = 5		k = 6		k = 7		k = 8	
	위양성 (비율)	위음성 (비율)	위양성 (비율)	위음성 (비율)	위양성 (비율)	위음성 (비율)	위양성 (비율)	위음성 (비율)
0.5	5502 (72.9%)	1 (0.1%)	6399 (92.2%)	0 (0.0%)	4740 (68.2%)	1 (0.1%)	5856 (84.3%)	0 (0.0%)
0.6	5502 (72.9%)	1 (0.1%)	2501 (36.0%)	1 (0.1%)	627 (9.0%)	2 (2.7%)	2224 (32.0%)	1 (0.1%)
0.7	355 (5.1%)	2 (0.2%)	40 (0.5%)	20 (2.7%)	627 (9.0%)	2 (2.7%)	101 (1.4%)	14 (1.9%)
0.8	355 (5.1%)	2 (0.2%)	40 (0.5%)	20 (2.7%)	23 (0.3%)	45 (6.2%)	18 (0.2%)	59 (8.1%)
0.9	16 (0.2%)	231 (32.0%)	15 (0.2%)	197 (27.3%)	15 (0.2%)	215 (29.8%)	13 (0.1%)	235 (32.6%)

실험 및 평가

- 신뢰도 평가

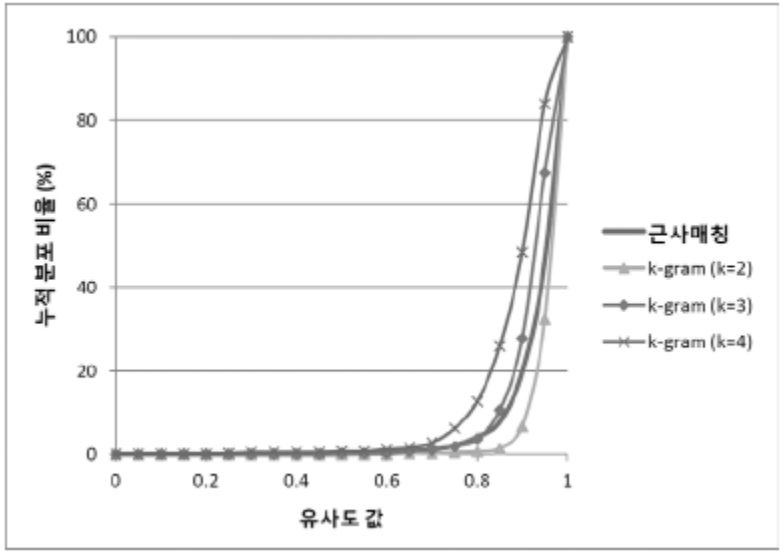
유사도 범위	근사 매칭		k-gram					
			k = 2		k = 3		k = 4	
	개수	비율	개수	비율	개수	비율	개수	비율
0-9	18,366	66.3	779	2.8	9,630	34.8	23230	84.0
10-19	5,670	20.4	2055	7.4	11,049	39.9	3804	13.8
20-29	2,207	7.9	5666	20.5	4,732	17.1	531	1.9
30-39	851	3.0	7984	28.9	1,689	6.1	58	0.2
40-49	372	1.3	5747	20.8	457	1.6	11	0.0
50-59	116	0.4	3184	11.5	75	0.2	13	0.0
60-69	43	0.1	1535	5.5	15	0.0	4	0.0
70-79	9	0.0	586	2.1	10	0.0	6	0.0
80-89	12	0.0	117	0.4	1	0.0	2	0.0
90-99	10	0.0	9	0.0	4	0.0	3	0.0
100	8	0.0	2	0.0	2	0.0	2	0.0
최소값	0.0		0.0		0.0		0.0	
최대값	100.0		100.0		100.0		100.0	
평균	9.3		37.6		14.9		5.5	
시간 (초)	13074.8		2839.7		3492.4		3834.0	



실험 및 평가

- 유사 프로그램에 대한 검출 능력 평가

유사도 범위	근사 매칭		k-gram					
			k = 2		k = 3		k = 4	
	개수	비율	개수	비율	개수	비율	개수	비율
0-9								
10-19								
20-29							2	0.4
30-39								
40-49					1	0.2		
50-59	2	0.4					3	0.6
60-69	3	0.6	1	0.2	3	0.6	7	1.6
70-79	13	2.9	2	0.4	12	2.7	42	9.6
80-89	68	15.5	26	6.0	105	24.0	157	36.0
90-99	332	76.1	394	90.4	310	71.1	220	50.5
100	18	4.1	13	2.9	5	1.1	5	1.1
최소값	54.6		62.5		48.3		21.0	
최대값	100.0		100.0		100.0		100.0	
평균	93.5		95.4		91.8		88.2	
시간 (초)	203.3		46.0		48.7		48.8	



k-gram의 근사 매칭을 이용한 간단한 실험

실험

- 같은 베이직 블록인지 판단하기 위한 실험
- 직접 구현한 max함수와 min함수의 블록을 가지고 실험


```
"mnemonics": ["push", "mov", "sub", "push", "push", "push", "lea", "mov", "mov", "stos", "mov", "cmp", "jle"],
"addr": 71792,
"number": 0,
"preds": [],
"succs": [71830, 71835],
"size": 38
```

```
"mnemonics": ["push", "mov", "sub", "push", "push", "push", "lea", "mov", "mov", "stos", "mov", "cmp", "jge"],
"addr": 71872,
"number": 0,
"preds": [],
"succs": [71910, 71915],
"size": 38
```

실험

```
class ngram(object):
    def __init__(self, mnemonics, n):
        self.ngramSet = []
        for i in range(len(mnemonics)):
            mnemonic = mnemonics[i:i+n]
            if len(mnemonic) != n:
                continue
            #delete duplicate n-gram
            if self.ngramSet.__contains__(mnemonic):
                continue
            self.ngramSet.append(mnemonic)
```

```
def _match(ngram1, ngram2, threshold):
    common = _lcs(ngram1, ngram2)
    common_len = float(len(common))
    match_score = (common_len*2/(len(ngram1)+len(ngram2)))
    if( match_score > threshold ):
        return True
    else:
        return False
```

```
def compare(P, Q, threshold):
    count = 0.0
    for ngram1 in P:
        for ngram2 in Q:
            if(_match(ngram1, ngram2, threshold)):
                count += 1
                break
    similarity = (count*2)/(len(P)+len(Q))
    print 'similarity ', similarity
    return similarity
```

실행코드

```
n1 = ngram(["push", "mov", "sub", "push", "push", "push", "lea", "mov", "mov", "stos", "mov", "cmp", "jle"], 5)
n2 = ngram(["push", "mov", "sub", "push", "push", "push", "lea", "mov", "mov", "stos", "mov", "cmp", "jge"], 5)
compare(n1.ngramSet, n2.ngramSet, 0.8)
```

결과

```
similarity  0.888888888889
```