



Information Hiding in Program Binaries

Rakan El-Khalil

xvr α xvr [.: net](http://xvr.net)

Warning:

- Steganography vs. Stenography.





Intro

- Information hiding overview
- Theoretical aspects of Software marking
- In practice...
- Applications



Types of Information Hiding

- Steganography
- Covert channels
- Anonymity
- Copyright marking
 - **Robust marks**
 - Fingerprinting
 - Watermarking [imperceptible or visible]
 - **Fragile marks**



General Methods

- Security through obscurity
 - **Mostly used historically**
 - **Sometimes used today**
- Camouflage
 - **Hiding in plain sight**
 - **Hiding the location of the embedded data**
- Spreading the hidden information

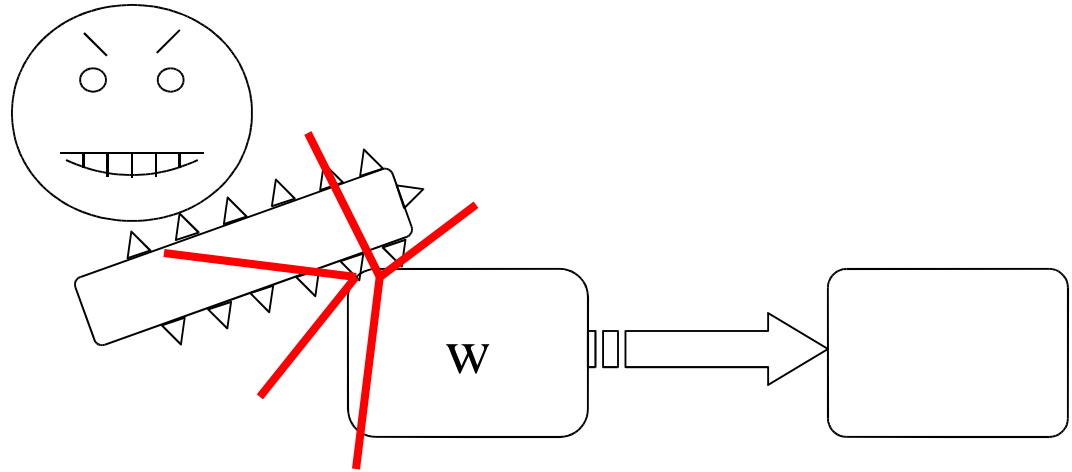


Strength Evaluation

- Data-Rate
- Stealth
- Resilience

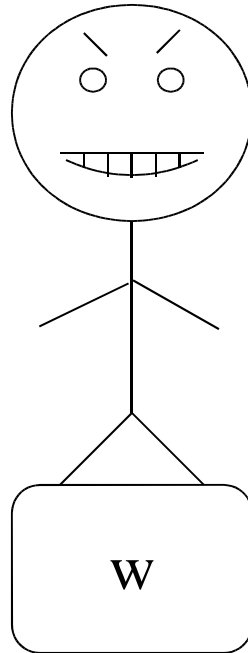
Attacks

- Subtractive
- Distortive
- Additive



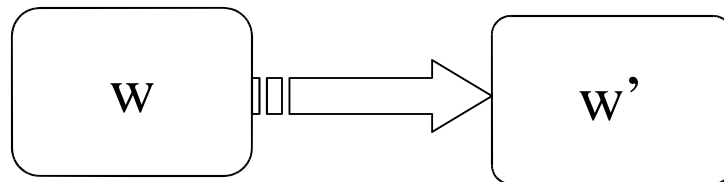
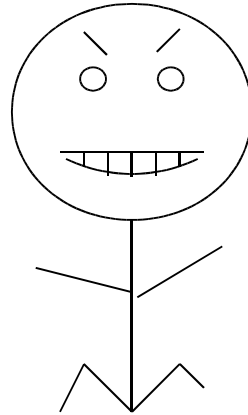
Attacks

- Subtractive
- Distortive
- Additive



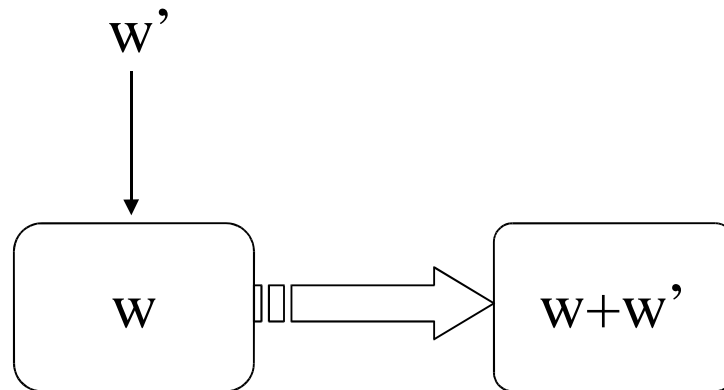
Attacks

- Subtractive
- Distortive
- Additive



Attacks

- Subtractive
- Distortive
- Additive





Mediums

- Sound
- Image
- Video
- Text
- Relational Databases
- Sets of Numbers
- Etc...



Binary Info Hiding Overview

- Low redundancy medium
- Static marks
- Dynamic marks

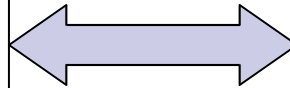
Static Data Marks

- `char mark[] = "All your base..."`
- ```
switch (a) {
 case 1: return "are" ;
 case 2: return "belong" ;
 case 3: return "to us" ;
 ...
}
```
- Etc...

# Static Code Marks

```
{
int gonads, strife;

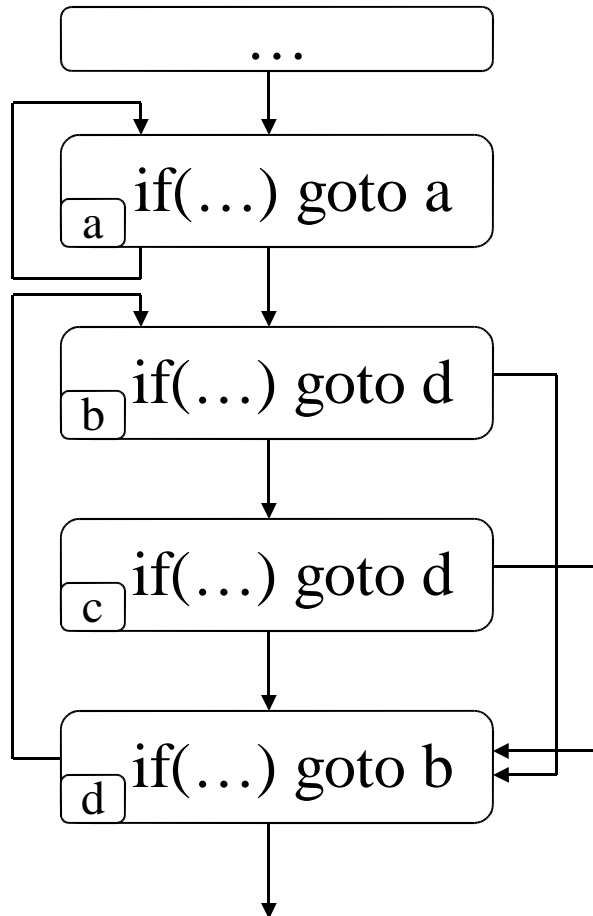
gonads = 1;
strife = 1;
printf ("weeeeeee");
}
```



```
{
int gonads, strife;

printf ("weeeeeee");
gonads = 1;
strife = 1;
}
```

# Static Code Marks [cont.]





# Static Code Marks [fin.]

- Pro: easy to implement
- Con: easy to break.

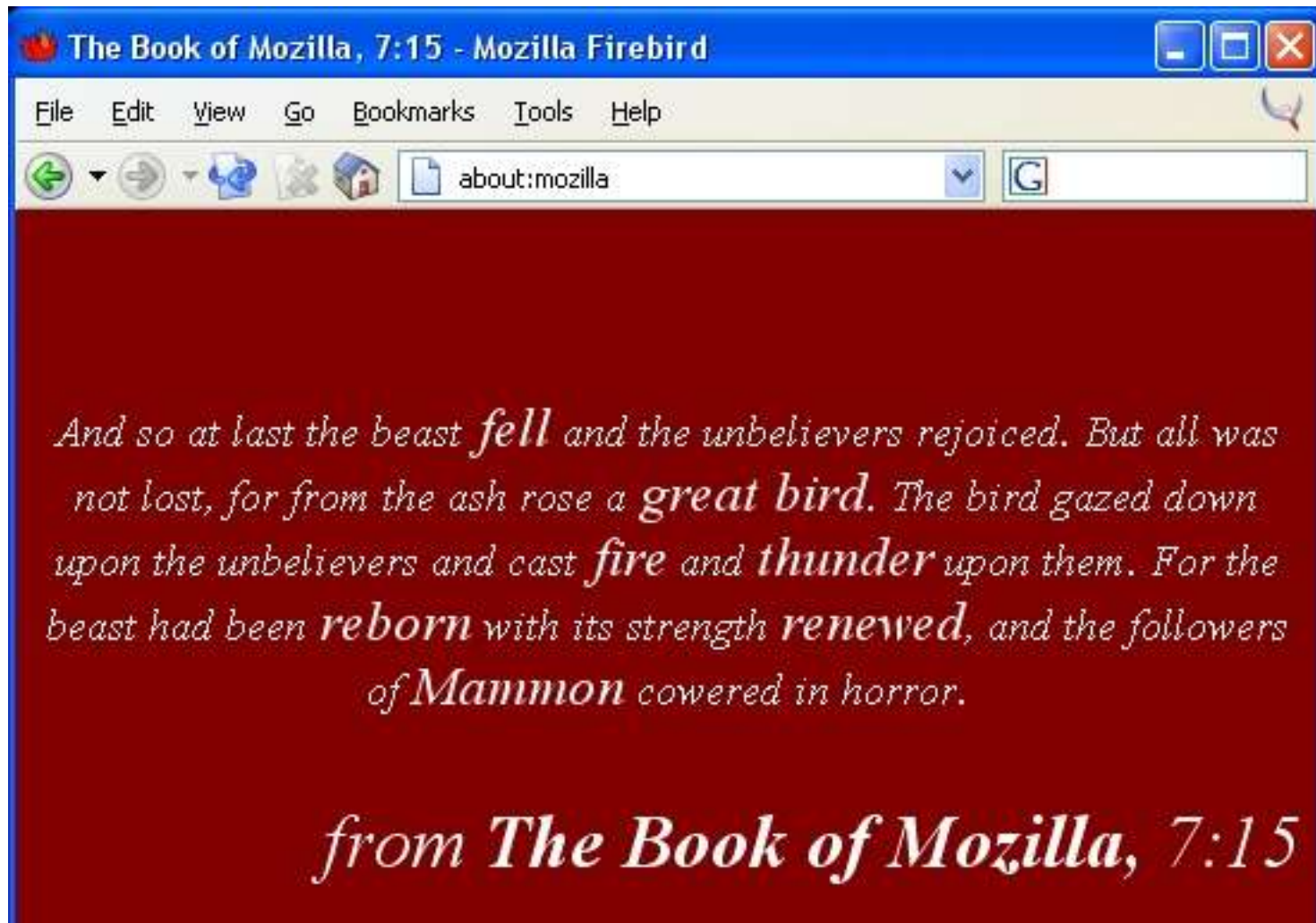




# Dynamic Marks

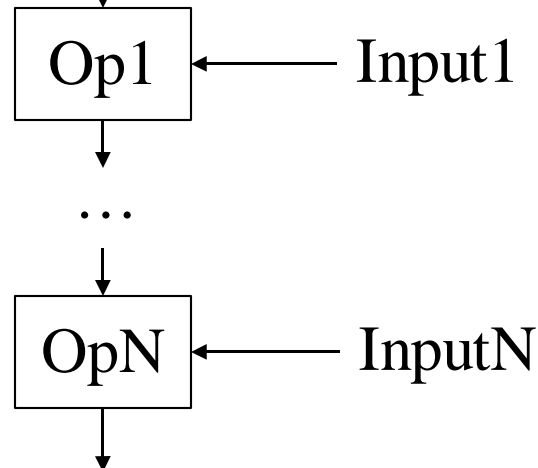
- Mark stored in program's execution state
- Types of marks:
  - **Data structure**
  - **Execution trace**
  - **Easter egg**

# Easter Egg



# Dynamic Data Structure

```
Var[0] = 0x01010101; Var[1] = 0x03030303;
Var[2] = 0x02020202; Var[3] = 0x04040404;
```

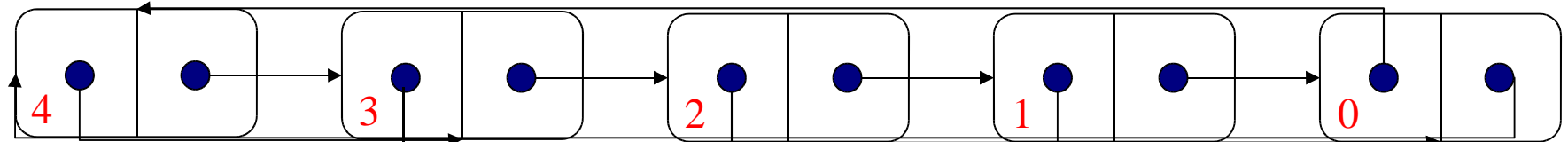


```
Var[0] = 0x54686520; Var[1] = 0x47726561;
Var[2] = 0x74204d61; Var[3] = 0x68697200;
```

“The Great Mahir”

# Dynamic Graph Watermarking

- Generate a number  $n = P \times Q$
- Watermark is the graph topology
- Eg: Radix-5 encoding



$$\begin{aligned} & 2 \cdot 5^4 + 0 \cdot 5^3 + 3 \cdot 5^2 + 2 \cdot 5^1 + 2 \cdot 5^0 \\ & = 7 \cdot 191 = 1337 \end{aligned}$$

# Dynamic Execution Trace

```
80480d3: 85 db
80480d5: 7e 29
80480d7: 83 7d 08 00
80480db: 74 23
80480dd: 8b 45 08
80480e0: a3 40 bc 08 08
80480e5: 80 38 00
 ...
 ...
8048100: b8 00 00 00 00
8048105: 85 c0
8048107: 74 0c
8048109: 83 c4 f4
```

```
 test %ebx,%ebx
 jle 0x8048100
 cmpl $0x0,0x8(%ebp)
 je 0x8048105
 mov 0x8(%ebp),%eax
 mov %eax,0x808bc40
 cmpb $0x0,(%eax)
 mov $0x0,%eax
 test %eax,%eax
 je 0x8048115
 add $0xffffffff4,%esp
```

# Attacks

- Semantics preserving transformations

```
{
 char c1, c2,
 c3;
 c1 = 'u';
 c2 = 'n';
 c3 = 'f';
}
```



```
char c1, c2, c3;
int i;
for (i=1; i <= 3; i++) {
 switch (i) {
 case 1:
 c1 = 'u' - 2; break;
 case 2:
 c2 = 'n' - 1;
 c1++; break;
 case 3:
 c3 = 'f';
 c1++; c2++; break;
 }}
}}
```



# Attacks on Dynamic Marking

- Add extra pointers
- Rename and reorder fields
- Add levels of indirection



# In Practice...

- Difference between bytecode [Java, .Net, etc] and machine code [x86 asm].
- Data vs. Code Problem.

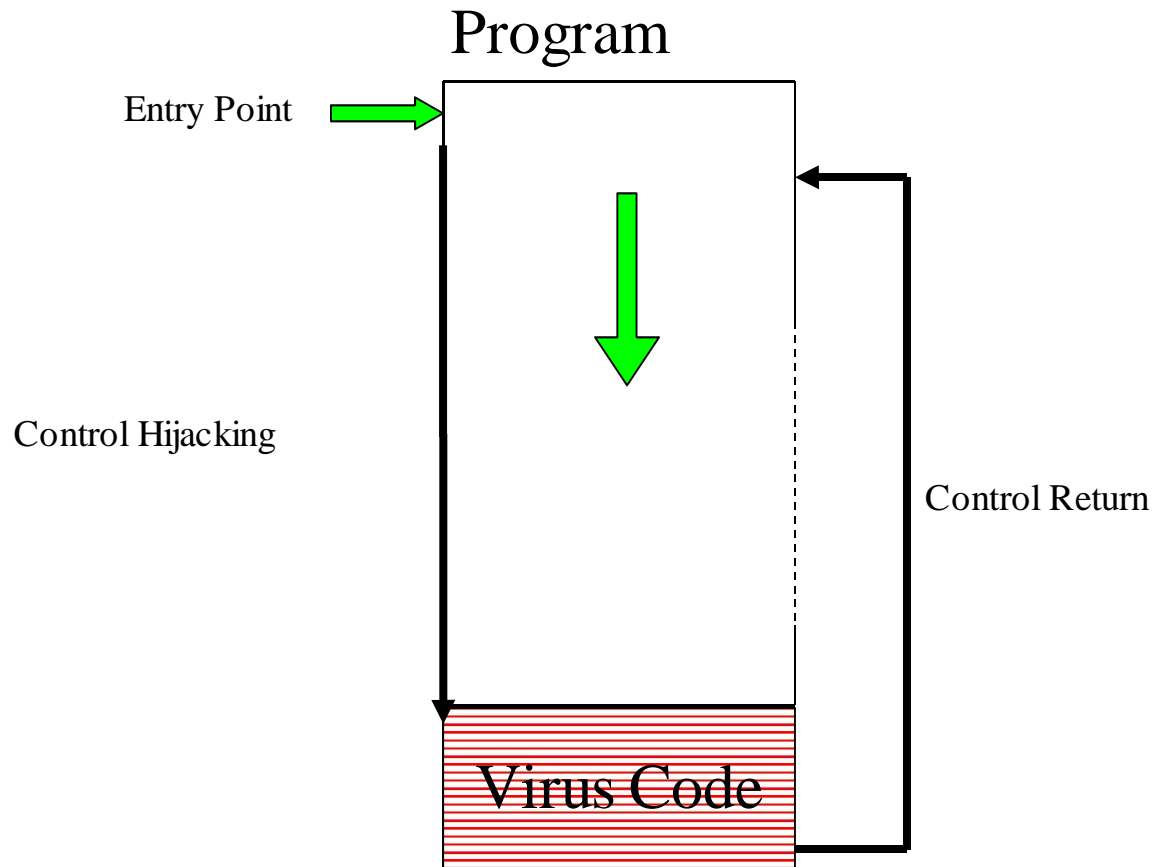




# In Practice

- Can't use advanced techniques.
- Little work done on machine code watermarking.

# Virus Intro





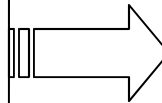
# Virus Code Obfuscation

- Encrypted
  - **Fixed decryption routine**
  - **Changing virus body**
- Polymorphic
  - **Mutation engine that randomizes decryption routine**
- Metamorphic
  - **No decryptor**
  - **Randomizes its code**

# Metamorphic Tricks

## ■ Register Swapping:

```
89 f6 mov %esi,%esi
80 38 2f cmpb $0x2f,(%eax)
75 09 jne 0x80480fa
8d 48 01 lea 0x1(%eax),%ecx
40 inc %eax
80 38 00 cmpb $0x0,(%eax)
```



```
89 c4 mov %eax,%eax
80 3e 2f cmpb $0x2f,(%esi)
75 09 jne 0x80480fa
8d 52 01 lea 0x1(%esi),%ebx
46 inc %esi
80 3e 00 cmpb $0x0,(%esi)
```



# More Tricks

- Instruction substitutions
- Changing of data values
- Nop and garbage insertions
- Branch reversals
- Alternate opcode encodings
- ...



# Hydan

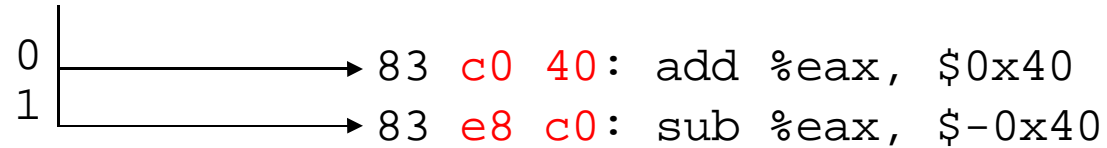
- Generic information hiding tool
- Works with instruction substitution



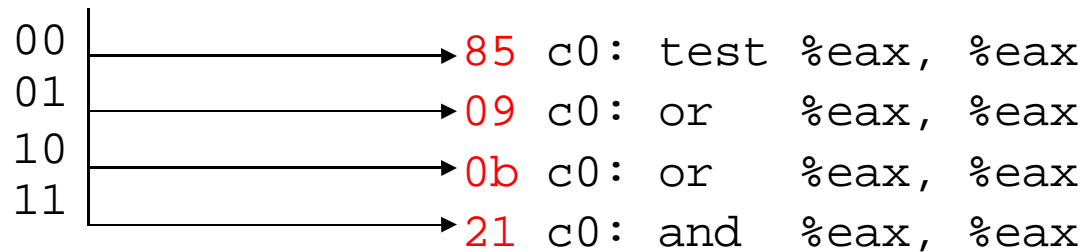
# Hydan Demo

# Example Substitutions

83 c0 40: add %eax, \$0x40



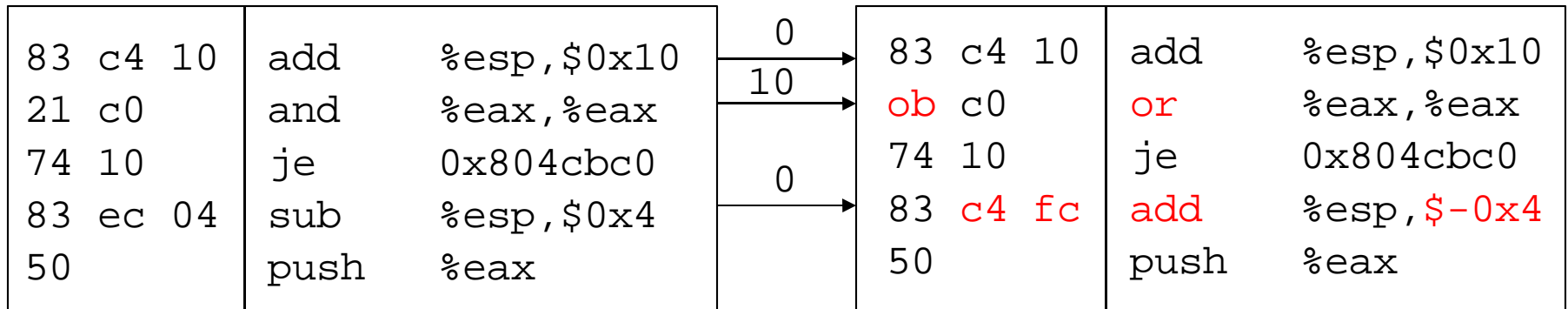
85 c0: test %eax, %eax





# Hydan Examples [cont.]

- Embed 0100 into a listing:

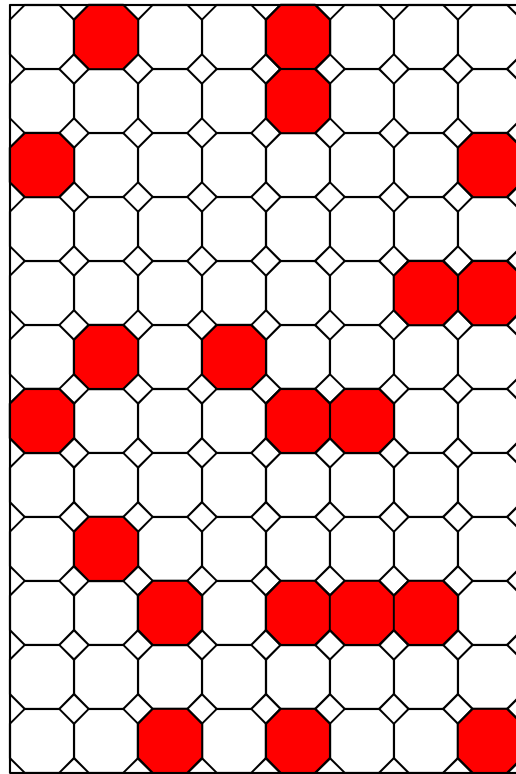




# Extra Security Techniques

- Message Whitening:
  - **ASCII text easily recognizable**
  - **Text encrypted with Blowfish in CBC mode**
  - **Length masked with SHA hash of password**

# Random Walk:



# Flag Collision Detection:

- Some 'equivalent' instructions set flags differently:
  - **Eg: add vs. sub**

```
xorl %eax, %eax
addl $0xffffffff, %eax

xorl %eax, %eax
subl $0x1, %eax
```

← %eax: -1, OF = 0, CF = 0

← %eax: -1, OF = 0, **CF = 1**

- What to do? Scan forwards.

# Codebook Embedding:

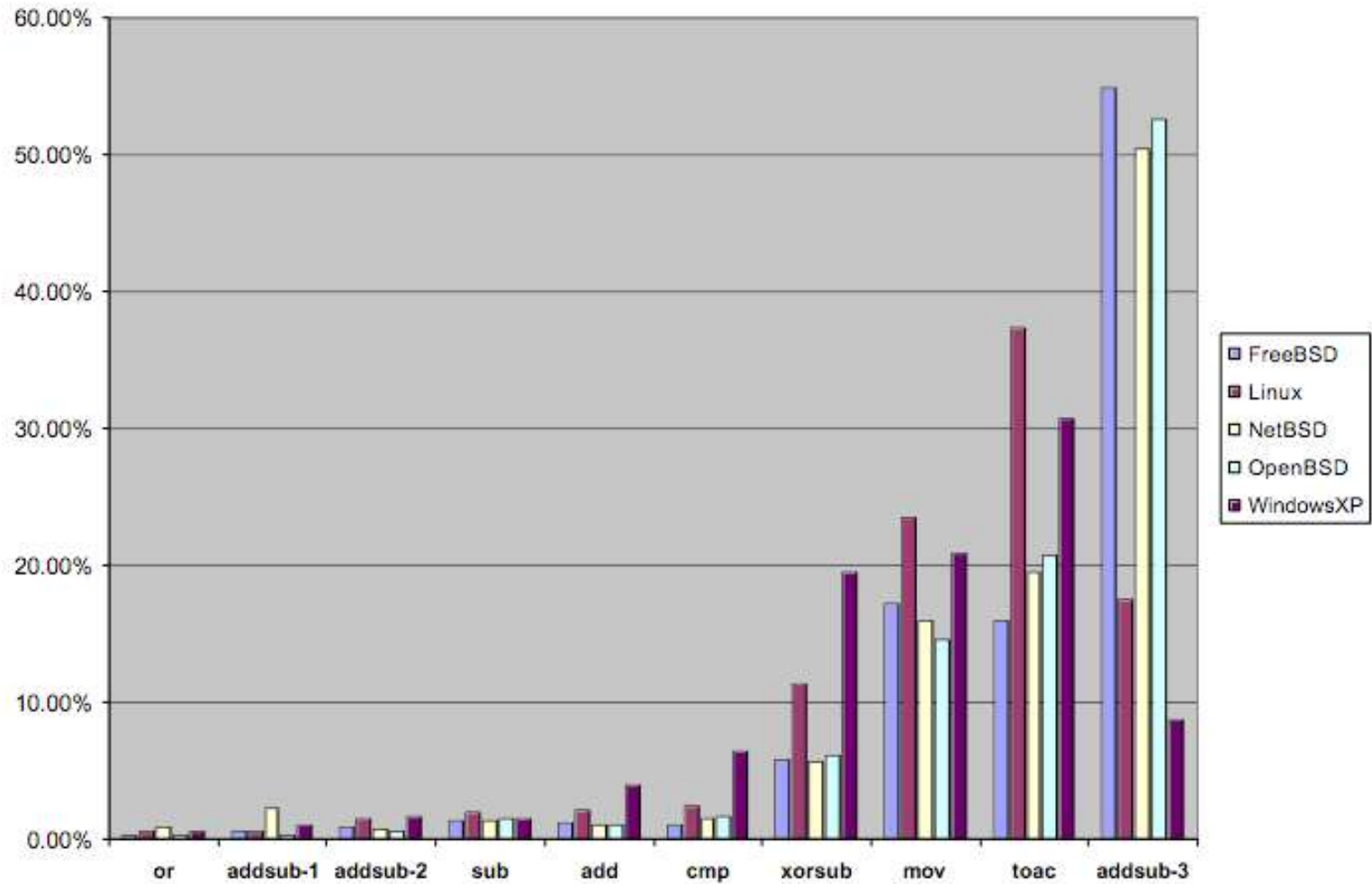
- Build a codebook of equivalent instructions:
  - **2 insns --> 1 bit**
  - **4 insns --> 2 bits**
  - **8 insns --> 3 bits**
- What happens if we have 7 insns?
- Encoding Rate:

$$\left\{ \begin{array}{l} \log_2(N): N \text{ is a power of } 2 \\ \log_2(N-1) \quad : \text{ otherwise} \end{array} \right.$$

# Hydan Issues

- Detectable
  - **Instructions are not created equal**
- Low bandwidth
  - **1/110 vs. Outguess' 1/17**
- Easy to tamper with
- Breaks SMC

# Statistics



# Future Work: Reordering

- Given a list of  $n$  objects:
  - **Can embed: floor [  $\log_2(n!)$  ] bits:**

| N  | Bits |
|----|------|
| 2  | 1    |
| 4  | 4    |
| 8  | 15   |
| 16 | 44   |
| 32 | 117  |
| 64 | 295  |





# Reordering Method

- How does one encode data with an ordering?
- Eg:  $n = 3$ :

000 → abc

001 → acb

010 → bac

011 → bca

100 → cab

101 → cba

# Encoding Algorithm

- More specifically:
  - **floor [  $\log_2(n!)$  ] bits of input**
  - **Take input and decompose it along its factorials.**
  - **Each factor represents the index of item in the sorted substring.**
- Eg:  $n = 4$ , input = 110110
  - **Floor [  $\log_2(4!)$  ]  $\rightarrow$  4bits**
  - **First 4 bits: 1101 == 13**
  - **Decomposition: 13 == 2\*3! + 0\*2! + 1\*1!**
  - **Resulting string: abcd  $\rightarrow$  cabd  $\rightarrow$  cabd  $\rightarrow$  cadb**

# Decoding Algorithm

- Similarly:
  - **floor [  $\log_2(\text{strlen}(\text{input})!)$  ] bits of output**
  - **Take input string and decompose it along its factorials.**
  - **Each factor represents the index of item in the sorted substring.**
- Eg: input = cadb
  - **Floor [  $\log_2(4!)$  ]  $\rightarrow$  4bits**
  - **Decomposition:  $2*3! + 0*2! + 1*1! == 13$**
  - **Result: 1101**



# Reorderables

- What can be reordered?
  - **Functions**
  - **Independent instructions**
  - **Arguments**
  - **Register allocation**
  - **Data**
  - **...**

# Instruction Reordering

```
mov $2, %eax
mov $3, %ebx
push %ecx
call $0x8056213
```

000

001

010

011

100

101

```
mov $2, %eax
mov $3, %ebx
push %ecx
```

```
mov $2, %eax
push %ecx
mov $3, %ebx
```

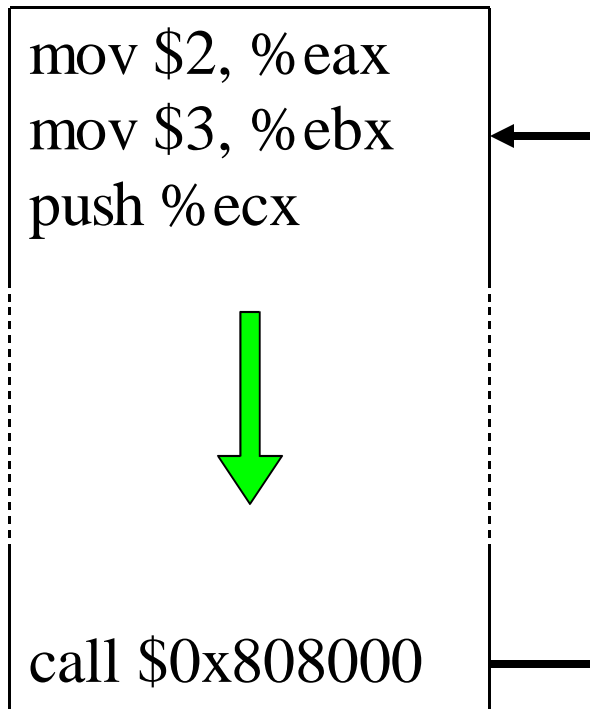
```
mov $3, %ebx
mov $2, %eax
push %ecx
```

```
mov $3, %ebx
push %ecx
mov $2, %eax
```

```
push %ecx
mov $2, %eax
mov $3, %ebx
```

```
push %ecx
mov $3, %ebx
mov $2, %eax
```

# Instruction Reordering Problem





# Insn Reordering Prob [cont]

- Need to use a VM to emulate:
  - **All Execution paths**
  - **Stack**
  - **Heap**





# Uses

- Traditional info hiding
- Security
- Polymorphism



# **Conclusion + QA**