Software-Based Approaches to Software Protection

Ginger Myles
mylesg@cs.arizona.edu

University of Arizona
Department of Computer Science
and
IBM Almaden Research Center
Introduction

In this talk we will discuss...
- The three major threats against IP in software
- Software-based solutions
3 Major Threats

- **Malicious Reverse Engineering**: the extracting of a piece of a program in order to reuse it in one's own.

- **Software Tampering**: the illegal modification of a program to circumvent licence checks, to obtain access to digital media protected by the software, etc.

- **Software Piracy**: the illegal reselling of legally obtained copies of a program.
Alice and Bob are competing software developers.

Bob reverse engineers Alice’s program and includes parts of it in his own code.

Easier with Java bytecode, .NET, . . .

⇒ Alice obfuscates her code.
What is Code Obfuscation?

A semantics-preserving transformation which
- makes the program harder to understand
- preserves original functionality
What is Code Obfuscation?

- A semantics-preserving transformation which
  - makes the program harder to understand
  - preserves original functionality

- Idea is to obscure **readability** and **understandability** to such a degree that it is **less costly** for the attacker to simply **recreate** the program or **purchase** a legal copy.
We want to develop an algorithm such that

- Maximize obscurity
- Maximize resilience
- Minimize cost
Code Obfuscation

- **Layout obfuscation:** alter the information that is unnecessary to the execution of the application such as identifier names and source code formatting.

- **Data obfuscation:** alter the data structures used by the program.

- **Control-flow obfuscation:** disguise the true control flow of the application.
Name obfuscation: rename the identifiers in the program to meaningless names.

```java
class C{
    void foo() {...}
    void bar(int i){...}
    void foobar(int i){...}
    String toString(){...}
}
```

```java
class C{
    void a() {...}
    void b(int i){...}
    void c(int i){...}
    String toString(){...}
}
```
**Name obfuscation:** rename the identifiers in the program to meaningless names.

```java
class C{
    void foo() { ... }
    void bar(int i) { ... }
    void foobar(int i) { ... }
    String toString() { ... }
}
```

```java
class C{
    void a() { ... }
    void a(int i) { ... }
    void b(int i) { ... }
    String toString() { ... }
}
```
Promote primitive types: change all primitives into instances of their respective wrapper classes.

```java
public class C {
    static int gcd(int x, int y) {
        int t;
        while (true) {
            boolean b = x % y == 0;
            if (b) return y;
            t = x % y;
            x = y;
            y = t;
        }
    }
    public static void main(String[] a) {
        System.out.println("Answer: ");
        System.out.println(gcd(100, 10));
    }
}
```

```java
public class C {
    static Integer gcd(Integer x, Integer y) {
        Integer t = null;
        while (true) {
            Boolean b = new Boolean(x.intValue() %
                                      y.intValue() == 0);
            if (b.booleanValue()) return y;
            t = new Integer(x.intValue() %
                             y.intValue());
            x = y;
            y = t;
        }
    }
    public static void main(String[] a) {
        System.out.println("Answer: ");
        System.out.println(gcd(new Integer(100),
                                new Integer(10)));
    }
}
```
Boolean splitter: every boolean variable is split into two variables and the state of the original variable is reflected in the combined state of the two variables.

```java
public class C {
    static int gcd(int x, int y) {
        int t;
        while (true) {
            boolean b = x % y == 0;
            if (b) return y;
            t = x % y; x = y; y = t;
        }
    }
    public static void main(String[] a) {
        System.out.print("Answer: ");
        System.out.println(gcd(100, 10));
    }
}
```

```java
public class C {
    static int gcd(int x, int y) {
        int t8, t7, t;
        for (; ;) {
            if (x%y==0) {
                t8=1; t7=0;
            } else {
                t8=0; t7=0;
            }
            if ((t7^t8)!=0)
                return y;
            else {
                t=x%y; x=y; y=t;
            }
        }
    }
    public static void main(String[] a) {
        System.out.print("Answer: ");
        System.out.println(gcd(100, 10));
    }
}
```
Control-flow Obfuscation

- Can be based on normal transformations an optimizing compiler would perform
  - method inlining
  - loop unrolling
Opaque Predicates

**Opaque Predicate:** A predicate $P$ is opaque at a program point $p$, if at point $p$ the outcome of $P$ is known at embedding time. If $P$ always evaluates to True we write $P^T_p$, for False we write $P^F_p$, and if $P$ sometimes evaluates to True and sometimes to False we write $P^?_p$.

Inserted to make it difficult for an adversary to analyze the control-flow of the application.
Sample Opaque Predicates

Number theoretically true opaque predicates

\[ \forall x, y \in \mathbb{Z} \quad 7y^2 - 1 \neq x^2 \]
\[ \forall x \in \mathbb{Z} \quad 2 \mid \lfloor \frac{x^2}{2} \rfloor \]
\[ \forall x \in \mathbb{Z} \quad 2 \mid x(x + 1) \]
\[ \forall x \in \mathbb{Z} \quad x^2 \geq 0 \]
\[ \forall x \in \mathbb{Z} \quad 3 \mid x(x + 1)(x + 2) \]
\[ \forall x \in \mathbb{Z} \quad 7 \nmid x^2 + 1 \]
\[ \forall x \in \mathbb{Z} \quad 81 \nmid x^2 + x + 7 \]
\[ \forall x \in \mathbb{Z} \quad 19 \nmid 4x^2 + 4 \]
\[ \forall x \in \mathbb{Z} \quad 4 \mid x^2(x + 1)(x + 1) \]
Control-flow Obfuscation

Bogus branch: use an opaque predicate to insert a bogus if statement.

```java
public class C {
    static int gcd(int x, int y) {
        int t;
        while (true) {
            boolean b = x % y == 0;
            if (b) return y;
            t = x % y; x = y; y = t;
        }
    }
    public static void main(String[] a){
        System.out.print("Answer: ");
        System.out.println(gcd(100,10));
    }
}
```

```java
public class C {
    static int gcd(int x, int y) {
        int t;
        int x = 2;
        while (true) {
            boolean b = x % y == 0;
            if (b) return y;
            t = x % y; x = y; y = t;
            if (x*x < 0) return 0;
        }
    }
    public static void main(String[] a){
        System.out.print("Answer: ");
        System.out.println(gcd(100,10));
    }
}
```
Code Obfuscation Example

Name obfuscation, boolean splitter, bogus branch, promote primitives

```java
public class C {
    static int gcd(int x, int y) {
        int t;
        while (true) {
            boolean b = x % y == 0;
            if (b) return y;
            t = x % y; x = y; y = t;
        }
    }

    public static void main(String[] a) {
        System.out.print("Answer: ");
        System.out.println(gcd(100, 10));
    }
}
```

```java
public class C {
    static Integer a(Integer x, Integer y) {
        Integer t8, t7, t;
        Integer b = new Integer(2);
        for (;;) {
            if (x.intValue() % y.intValue() == 0) {
                t8 = new Integer(1); t7 = new Integer(0);
            } else {
                t8 = new Integer(0); t7 = new Integer(0);
                if (((t7.intValue()) ^ t8.intValue()) != 0) {
                    return y;
                } else {
                    t = new Integer(x.intValue() %
                        y.intValue());
                    x = y; y = t;
                }
                if (b.intValue() * b.intValue() < 0)
                    return 0;
            }
        }
    }

    public static void main(String[] a) {
        System.out.print("Answer: ");
        System.out.println(a(new Integer(100),
            new Integer(10)));
    }
}
```
Alice is software developer.

Alice gives Bob a trial version of her software.

Bob locates the expiration check and disables it. He now has unlimited use of the software.

⇒ Alice tamperproofs her program.
What is Software Tamperproofing?

- Code obfuscation is used to hide a secret.
- Tamperproofing is used to protect the secret from alteration.
What is Software Tamperproofing?

- Code obfuscation is used to **hide** a secret.
- Tamperproofing is used to **protect** the secret from **alteration**.
- Tamperproofing performs two duties
  1. Detect that the software has been altered.
  2. Once detection has occurred, cause the program to fail or repair itself.
We want to design an algorithm such that
  - software failure is stealthy
  - does not alert the attacker to the location of the failure inducing code.
We want to design an algorithm such that software failure is stealthy and does not alert the attacker to the location of the failure inducing code.

```markdown
if (tampering)
    abort();
```
Software Tamperproofing

- Tamper detection techniques
  - Inspect the code
  - Inspect the state
  - Generate code
Software Tamperproofing

- Tamper detection techniques
  - Inspect the code
  - Inspect the state
  - Generate code
- No matter which technique, the expected result should not be revealed to the attacker.

```c
if (currentDate >= Feb 9, 2006)
    exit();
```
Key is the integrity verification kernels.

Each IVK contains $2^n$ blocks of code.

Half are in upper memory, half in lower memory.

XOR each block in upper memory with a block in lower memory.

Result one block is decrypted and execution resumes at that block.
Alice is a software developer.

Bob buys one copy of Alice’s application and sells copies to third parties.

⇒ Alice watermarks/fingerprints her program.
What is Software Watermarking?

- A technique used to aid in the prevention of software piracy.
- Embed a **unique identifier** in a program.

**Watermarking**
- Same identifier
- Copyright notice
- Discourages theft

**Fingerprinting**
- Different identifier
- Customer identification
- Trace illegal copies

- Discourages but does **not** prevent illegal copying and redistribution.
A watermarking system consists of two functions:

- \( \text{embed}(P, w, key) \rightarrow P' \)
- \( \text{recognize}(P', key) \rightarrow w \)
We want to develop an algorithm such that when we embed the watermark $W$ in the program $P$

- $W$ is resilient to various attacks.
- $W$ is stealthy.
- $W$ is large (high bit-rate).
- The overhead (space and time) is low.
Subtractive Attack: The adversary examines the (disassembled/de-compiled) program in an attempt to discover the watermark and to remove all or part of it from the code.
Additive Attack: The adversary adds a new watermark in order to make it hard for the IP owner to prove that her watermark is actually the original.
Attacks on Software Watermarks

**Distortive Attack:** A series of semantics-preserving transformations are applied to the software in an attempt to render the watermark useless.
Attacks on Software Watermarks

**Collusive Attack:** The adversary compares two copies of the software which contain different fingerprints in order to identify the location.
Naive Watermarking Techniques

Constant String

```java
String watermark = "Copyright 2006 ...";
```

```java
String watermark = "CC Number 1234 ...";
```

- Easy to attack, unstealthy, high bit-rate, little overhead.
Naive Watermarking Techniques

Switch Encoding

```c
switch (E) {
    case 1: {⋯}
    case 5: {⋯}
    case 9: {⋯}
}
```

```c
switch (E) {
    case 5: {⋯}
    case 1: {⋯}
    case 9: {⋯}
}
```

- Easy to attack, **stealthy**, low bit-rate, no overhead.
Naive approaches:

- Renaming

![Diagram showing watermark embedding process]

\[ P \xrightarrow{L} \text{Embed} \xrightarrow{X} P' \]
Watermarking Transformations

Naive approaches:

- Renaming

- Reordering
Advanced approaches:

- Alter program statistics

![Diagram showing watermarking transformations with a process labeled 'Embed']
Watermarking Transformations

Advanced approaches:

- Alter program statistics

- Extend program semantics
Monden et al.

- Semantics extending transformation.
- Embeds the watermark in a dummy method that is added to the application.
- The embedding is accomplished through a specially constructed sequence of instructions.
- Since the inserted method is never executed there is flexibility in how the instructions are constructed.
- Can disguise the method by adding a call to the method which is regulated by an opaque predicate.

COMPSAC 2000.
Encode 8 bits of the watermark by replacing the operand of every BIPUSH instruction.

```java
void whileI() {
    int i = 0;
    while (i < 100) {
        i++;  
    }
}
```

0  `iconst_0`  // push int constant 0  
1  `istore_1`  // store into local variable 1  
2  `goto 8`   // first time no increment  
5  `iinc 1 1` // add 1 to local variable 1  
8  `iload_1`  // load from local variable 1  
9  `bipush 100` // push a small int (100)  
11 `if_icmplt 5` // compare, if true goto 5  
14 `return` // return void when done
Monden et al.

Encode 3 bits of the watermark by replacing each arithmetic instruction.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>iadd</td>
<td>000</td>
</tr>
<tr>
<td>iand</td>
<td>001</td>
</tr>
<tr>
<td>ior</td>
<td>010</td>
</tr>
<tr>
<td>ixor</td>
<td>011</td>
</tr>
<tr>
<td>irem</td>
<td>100</td>
</tr>
<tr>
<td>idiv</td>
<td>101</td>
</tr>
<tr>
<td>imul</td>
<td>110</td>
</tr>
<tr>
<td>isub</td>
<td>111</td>
</tr>
</tbody>
</table>

```
int fact(int x){
    int factorial = 1;
    for(int i=1; i <= x; i++){
        factorial *= i;
    }
    return factorial;
}
```

```
0  iconst_1
1  istore_1
2  iconst_1
3  istore_2
4  goto 14
7  iload_1
8  iload_2
9  imul
10 istore_1
11 iinc 2 1
14 iload_2
15 iload_0
16 if_icmple 7
19 iload_1
20 ireturn
```
Semantics extending transformation.

$k$ branching points throughout the application are randomly selected.

At each branching point either $\land P^T$, $\lor \neg P^T$, or $\lor P^F$ is appended to the predicate at that location.

The bits of the watermark are embedded through the opaque predicate that has been chosen.

Within the opaque predicate the bits can be encoded either as constants or by assigning a rank to each of the opaque predicates.

ICECR-5, 2002.
Arboit Algorithm 1 Example

class C{
    void m1(int a, int b) {
        ...
        if (a <= b) {...}
        else {...}
        ...
    }
}

class C{
    void m1(int a, int b) {
        ...
        int c = 1;
        if ((a <= b) && (c*c >= 0)) {...}
        else {...}
        ...
    }
}
Qu and Potkonjak

Renaming transformation.

Embed the mark by adding constraints (extra edges) to the register interference graph.

Easy to attack by random register re-numbering.

Interference Graph

- Models the relationship between the variables in the procedure.
- Each variable in the procedure is represented by a vertex.
- If two variables have overlapping live ranges then the vertices are joined by an edge.
- The graph is colored so that we can assign the variables to registers so that we minimize the number of registers required and variables that are live at the same time do not share a register.
Interference Graph Example

v1 := 2 * 2
v2 := 2 * 3
v3 := 2 * v2
v4 := v1 + v2
v5 := 3 * v3
Interference Graph Example

\begin{align*}
v1 & := 2 \times 2  
v2 & := 2 \times 3  
v3 & := 2 \times v2  
v4 & := v1 + v2  
v5 & := 3 \times v3 \end{align*}
\[v_1 := 2 \times 2\]
\[v_2 := 2 \times 3\]
\[v_3 := 2 \times v_2\]
\[v_4 := v_1 + v_2\]
\[v_5 := 3 \times v_3\]
(a) Original Bytecode

```
METHOD: fast_memcmp:([B[BI)Z
0 : iconst_0
1 : istore 3
3 : iconst_0
4 : istore_3
5 : iconst_1
6 : istore 3
8 : iload_2
9 : iconst_1
10 : isub
11 : istore_2
12 : iload_2
13 : iconst_0
14 : if_icmplt -> 21
17 : iconst_0
18 : goto -> 22
21 : iconst_1
22 : ifne -> 33
23 : iload 3
25 : iload 3
27 : invokesub
30 : goto -> 34
33 : iconst_1
34 : if_icmpeq -> 50
37 : iload_2
38 : iload_2
40 : iload_1
41 : iload_2
42 : iload
43 : if_icmpeq -> 50
46 : iconst_0
47 : goto -> 51
50 : iconst_1
51 : istore 3
53 : iload_2
54 : iconst_1
55 : isub
56 : istore_2
57 : goto -> 12
60 : iload 3
62 : ireturn
```

(b) Original Interference Graph

(c) Watermarked Interference Graph

(d) Register Assignment Table

<table>
<thead>
<tr>
<th>variable</th>
<th>register number</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>0</td>
</tr>
<tr>
<td>v2</td>
<td>1</td>
</tr>
<tr>
<td>v3</td>
<td>2</td>
</tr>
<tr>
<td>v4</td>
<td>3</td>
</tr>
<tr>
<td>v5</td>
<td>3</td>
</tr>
<tr>
<td>v6</td>
<td>4</td>
</tr>
<tr>
<td>v7</td>
<td>2</td>
</tr>
</tbody>
</table>

(e) Watermarked Bytecode

```
METHOD: fast_memcmp:([B[BI)Z
0 : iconst_0
1 : istore 3
3 : iconst_0
4 : istore_3
5 : iconst_1
6 : istore 3
8 : iload_2
9 : iconst_1
10 : isub
11 : istore_2
12 : iload_2
13 : iconst_0
14 : if_icmplt -> 21
17 : iconst_0
18 : goto -> 22
21 : iconst_1
22 : ifne -> 33
23 : iload 3
25 : iload 3
27 : invokesub
30 : goto -> 34
33 : iconst_1
34 : if_icmpeq -> 60
37 : iload_2
38 : iload_2
40 : iload_1
41 : iload_2
42 : iload
43 : if_icmpeq -> 50
46 : iconst_0
47 : goto -> 51
50 : iconst_1
51 : istore 4
53 : iload_2
54 : iconst_1
55 : isub
56 : istore_2
57 : goto -> 12
60 : iload 4
62 : ireturn
```
A technique used to address the illegal distribution of all or some part of a program.

Extract identifying characteristics from two programs to show that one is a copy of the other.
How Does Birthmarking Differ From Watermarking?

- It is often necessary to alter existing code or add code to the application in order to embed the watermark.
- Birthmarks **cannot** prove authorship or identify the source of an illegal redistribution.
- Birthmarks only **confirm** that one program is a copy of another.
A Birthmarking system consists of the following functions:

- $\text{extract}(p) \rightarrow b_p$
- $\text{extract}(q) \rightarrow b_q$
- $\text{similarity}(p, q) \rightarrow [0, 1]$
A Birthmarking system consists of the following functions:

- extract(p) → \( b_p \)
- extract(q) → \( b_q \)
- similarity(p, q) → \([0, 1]\)

We want to develop an algorithm such that

- \( B \) is resilient to various transformations.
- \( B \) is credible.
Software Birthmarking

- Idea similar to that of:
  - Plagiarism detection
  - Code clones
- What makes it unique is that a birthmark is computed at the machine code level and considers semantics-preserving transformations.
Compute the set of unique opcode sequences of length $k$ for a set of modules.

$k = 3$ minimized the probability of false positives while maximizing resistance to transformations.

ACM SAC 2005.
Myles and Collberg

k-gram where $k = 2$

Method `void main(java.lang.String [])`

```
0 getstatic #13 <Field java.io.PrintStream out>
3 new #16 <Class java.lang.StringBuffer>
6 dup
7 ldc #18 <String "15!==">
9 invokespecial #23 <Method java.lang.StringBuffer(java.lang.String)>
12 ldc2_w #24 <Long 15>
15 invokestatic #29 <Method long fact(long)>
18 invokevirtual #33 <Method java.lang.StringBuffer append(long)>
21 invokevirtual #37 <Method java.lang.String toString()>
24 invokevirtual #40 <Method void println(java.lang.String)>
27 return
```

{(getstatic, new),
(new, dup),
(dup, ldc),
(ldc, invokespecial),
(invokespecial, ldc2_w),
(ldc2_w, invokestatic),
(invokestatic, invokevirtual),
(invokervirtual, invokevirtual),
(invokervirtual, return)}
Computing similarity:

\[
similarity(b_p, b_q) = \frac{|b_p \cap b_q|}{|b_p|}
\]
Birthmarks and Watermarks

- Birthmarks provide weaker evidence than software watermarks. They can indicate that one program is likely to be a copy of another, but not who the original author is or who is guilty of piracy.
- Birthmarks can be used in instances where watermarking is not feasible.
- Birthmarks can be used in conjunction with watermarking to provide stronger evidence of theft.
- There are instances where watermarks are destroyed but birthmarks are not.
Summary

The IP in software is threatened in several different ways.

There are a variety of techniques to address the different issues.

The goal behind software-based techniques is to require “enough” time, effort, and/or resources to break such that it is less costly for the attacker to simply rewrite the software or purchase legal copies.

Currently no single mechanism exists to prevent all three threats.

By combining the techniques, a stronger defense which provides multiple levels of protection can be achieved.
Questions?
Arboit Algorithm 2

- \( k \) branching points throughout the application are randomly selected.
- At each branching point, \( M^T_{bi} \) or \( M^F_{bi} \) is created and a method call is appended.
- The bits of the watermark are encoded in the opaque method through the opaque predicate that it evaluates.
class C{
    void m1(int a, int b){
        ...
        if (a <= b){...}
        else {...}
        ...
    }
}

class C{
    boolean m2(){
        int c = 1;
        return (c*c >= 0);
    }
    void m1(int a, int b){
        ...
        if ((a <= b) && m2()) {...}
        else {...}
        ...
    }
}
Tamada et al.

- Specific to Java class files.
- Composed of 4 individual birthmarks
  - constant value in field variables
  - sequence of method calls
  - inheritance structure
  - used classes
Constant value in field values:

```java
public class SkinPanel extends JPanel
    implements SandMarkGUIConstants
{

    private Image image;
    private int imgWidth = -1;
    private int imgHeight = -1;

    public SkinPanel(){
        setLayout(null);
        setBackground(new Color(0xe8d5bd));
        image = ToolKit.getDefaultToolkit().getImage(
            getClass().getClassLoader().getResource(SAND_IMAGE));
        if(image != null){
            MediaTracker med = new MediaTracker(this);
            med.addImage(image, 0);
            try{
                med.waitForAll(10000);
            } catch(Exception e){
                throw new RuntimeException(ex);
            }
            while((imgWidth = image.getWidth(null)) == -1 ||
                (imgHeigth = image.getHeight(null)) == -1);
        }
    }
}
```
Sequence of method calls:

```java
public class SkinPanel extends JPanel implements SandMarkGUIConstants {
    private Image image; int imgWidth = -1; int imgHeight = -1;
    public SkinPanel()
    {
        setLayout(null); setBackground(new Color(0xe8d5bd));
        image = Toolkit.getDefaultToolkit().getImage(getClass().getClassLoader().getResource(SAND_IMAGE));
        if(image != null){
            MediaTracker med = new MediaTracker(this); med.addImage(image, 0);
            try{
                med.waitForAll(10000);
            }catch(Exception e){
                throw new RuntimeException(e);
            }
            while((imgWidth = image.getWidth(null)) == -1 || (imgHeight = image.getHeight(null)) == -1);
        }
    }
}
```
Inheritance structure:

```java
public class SkinPanel extends JPanel
    implements SandMarkGUIConstants
{

    private Image image;
    private int imgWidth = -1;
    private int imgHeight = -1;

    public SkinPanel()
    {
        setLayout(null);
        setBackground(new Color(0xe8d5bd));
        image = ToolKit.getDefaultToolKit().getImage(
            getClass().getClassLoader().getResource(SAND_IMAGE));
        if (image != null)
        {
            MediaTracker med = new MediaTracker(this);
            med.addImage(image, 0);
            try{
                med.waitForAll(10000);
            } catch(Exception e){
                throw new RuntimeException(ex);
            }
            while((imgWidth = image.getWidth(null)) == -1 ||
                (imgHeight = image.getHeight(null)) == -1);
        }
    }
}
```
Used classes:

```java
public class SkinPanel extends JPanel
    implements SandMarkGUIConstants
{

    private Image image;
    private int imgWidth = -1;
    private int imgHeight = -1;

    public SkinPanel()
    {
        setLayout(null);
        setBackground(new Color(0xe8d5bd));
        image = ToolKit.getDefaultToolkit().getImage(
            getClass().getClassLoader().getResource(SAND_IMAGE));
        if (image != null)
        {
            MediaTracker med = new MediaTracker(this);
            med.addImage(image, 0);
            try{
                med.waitForAll(10000);
            } catch(Exception e){
                throw new RuntimeException(ex);
            }
            while((imgWidth = image.getWidth(null)) == -1 ||
                (imgHeight = image.getHeight(null)) == -1);
        }
    }
}
```

java.awt.Color,
java.awt.Component,
java.awt.Image,
java.awt.image.ImageObserver,
java.awt.MediaTracker,
java.awt.Toolkit,
java.lang.Class,
java.lang.ClassLoader,
java.lang.Object,
java.lang.RuntimeException,
java.lang.String,
java.lang.Throwable,
java.net.URL,
javax.swing.JPanel
Computing similarity:

- CVFV: $\text{similarity}(b_p, b_q) = \frac{|b_p \cap b_q|}{|b_p|}$

- SMC, IS, UC: $\text{similarity}(b_q, b_q) = \frac{|LCS(b_p, b_q)|}{|b_p|}$