Recognizing and Imitating Programmer Style: Adversaries in Program Authorship Attribution

Lucy Simko, Luke Zettlemoyer, Tadayoshi Kohno

simkol@cs.washington.edu
homes.cs.washington.edu/~simkol
int main()
{
    int i, j, k, l, m, n, st;
    char in[10000];
    int fg[5000], chk[128];
    int size, count = 0, res;
    scanf("%d%d%d", &len, &n, &size);
    rep (i, n) scanf("%s", dic[i]);

    while (size--)
    {
        scanf("%s", in);
        st = 0;
        rep (k, n) fg[k] = 1;
        ...
    }
}

- 98% accuracy over 250 programmers
- Extract syntactic, lexical, and layout features from C/C++ code
- Random Forest classifier
- Data set: Google Code Jam
  - Programming competition
  - Lots of examples of people solving the same problem in different ways
- Open source
int main()
{
    int i, j, k, l, m, n, st;
    char in[10000];
    int fg[5000], chk[128];
    int size, count = 0, res;
    scanf("%d%d%d", &len, &n, &size);
    rep (i, n) scanf("%s", dic[i]);

    while (size--)
    {
        scanf("%s", in);
        st = 0;
        rep (k, n) fg[k] = 1;
        ...
    
}
int main()
{
    int i, j, k, l, m, n, st;
    char in[10000];
    int fg[5000], chk[128];
    int size, count = 0, res;
    scanf ("%d%d%d", &len, &n, &size);
    rep (i, n) scanf ("%s", dic[i]);

    while (size--)
    {
        scanf ("%s", in);
        st = 0;
        rep (k, n) fg[k] = 1;
        ...
    }
}
int main()
{
    int i, j, k, l, m, n, st;
    char in[10000];
    int fg[5000], chk[128];
    int size, count = 0, res;
    scanf ("%d%d%d", &len, &n, &size);
    rep (i, n) scanf ("%s", dic[i]);
    
    while (size--)
    {
        scanf ("%s", in);
        st = 0;
        rep (k, n) fg[k] = 1;
        
        ...
Can we fool source code attribution classifiers?

Yes!

Methodology: Lab study* with C programmers

*Approved by University of Washington’s Human Subjects Division (IRB)
Outline

- Motivation and Research Question
- Source Code Attribution: Overview and Background
- Evading Source Code Attribution: Definitions and Goals
- Methodology
- Results: Conservative Estimate of Adversarial Success
- Results: How to Create Forgeries
int main()
{
    int i, j, k, l, m, n, st;
    char in[10000];
    int fg[5000], chk[128];
    int size, count = 0, res;
    scanf("%d%d%d", &len, &n, &size);
    rep (i, n) scanf("%s", dic[i]);

    while (size--)
    {
        scanf("%s", in);
        st = 0;
        rep (k, n) fg[k] = 1;
        ...
    }
int main()
{
    int i, j, k, l, m, n, st;
    char in[10000];
    int fg[5000], chk[128];
    int size, count = 0, res;
    scanf("%d%d%d", &len, &n, &size);
    rep (i, n) scanf("%s", dic[i]);
    while (size--)
    {
        scanf("%s", in);
        st = 0;
        rep (k, n) fg[k] = 1;
        ...
int main()
{
    int i, j, k, l, m, n, st;
    char in[10000];
    int fg[5000], chk[128];
    int size, count = 0, res;
    scanf("%d%d%d", &len, &n, &size);
    rep(i, n) scanf("%s", dic[i]);
    while (size--)
    {
        scanf("%s", in);
        st = 0;
        rep(k, n) fg[k] = 1;
        ...
    }
Source Code Attribution

$P_c \rightarrow \text{Classifier} \rightarrow \{A, B, C, D, E\}$
Source Code Attribution

$P_c \xrightarrow{} \text{Classifier} \xrightarrow{} C$

\{A, B, C, D, E\}

Who the classifier thinks wrote this code.
Motivation and Research Question

Source Code Attribution: Overview and Background

Evading Source Code Attribution: Definitions and Goals

Methodology

Results: Conservative Estimate of Adversarial Success

Results: How to Create Forgeries
1. **Train**: Given code from *original* and *target authors*, learn styles
2. **Modify original code** to imitate target author (**forgery**)
   - Or just hide the original author’s style (**masking**)

---

**Evading Source Code Attribution**

\[ P_c \xrightarrow{\text{Adversarial manipulation}} P_{c'} \]

- Code originally by C, but modified by an adversary.
1. **Train**: Given code from **original and target authors**, learn styles
2. **Modify original code** to imitate target author (**forgery**)  
   - Or just hide the original author’s style (**masking**)
Outline

- Motivation and Research Question
- Source Code Attribution: Overview and Background
- Evading Source Code Attribution: Definitions and Goals
- **Methodology**
- Results: Conservative Estimate of Adversarial Success
- Results: How to Create Forgeries
Lab Study: Dataset

- C code
- We used a linter\(^1\) to eliminate many typographic style differences
- ~4000 authors: avg 2.2 files each
- 5 authors with the most files: avg ~42.8 files
  - Authors: A, B, C, D, E

\(^1\) http://astyle.sourceforge.net/
Lab Study: Create Forgeries

Precision: 100%
Recall: 100%
(10-fold XV)

$C_5$

{A, B, C, D, E}
Lab Study: Create Forgeries

Precision: 87.6%
Recall: 88.2%
(10-fold XV)

$C_{20}$

{A, B, C, D, E, ... + 15}
Lab Study: Create Forgeries

$C_{50}$

Precision: 82.3%
Recall: 84.5%
(10-fold XV)

$\{A, B, C, D, E, \ldots + 45\}$
Lab Study: Create Forgeries

28 C programmers (participants):

1. **Train:** Given code from *original and target author*, learn styles
2. **Modify original code** to imitate target author’s style (*forgery*)

\[ P_x, P_x', \text{ Classifier} \rightarrow Y \]

\[ X, Y \in \{A, B, C, D, E\} \]
Lab Study: Create Forgeries

28 C programmers (participants):

1. **Train**: Given code from **original and target author**, learn styles
2. **Modify original code** to imitate target author’s style **(forgery)**
3. **Check forgery success** against oracle classifiers

\[ X, Y \in \{A, B, C, D, E\} \]
Outline

- Motivation and Research Question
- Source Code Attribution: Overview and Background
- Evading Source Code Attribution: Definitions and Goals
- Methodology
  - Results: Conservative Estimate of Adversarial Success
  - Results: How to Create Forgeries
Results: Estimate of Adversarial Success

Versions of the state-of-the-art machine classifier. The subscript indicates the number of authors in the training set.

<table>
<thead>
<tr>
<th></th>
<th>$C_5$</th>
<th>$C_{20}$</th>
<th>$C_{50}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forgery</td>
<td>66.6%</td>
<td>70.0%</td>
<td>73.0%</td>
</tr>
<tr>
<td>Masking</td>
<td>76.6%</td>
<td>76.6%</td>
<td>86.6%</td>
</tr>
</tbody>
</table>

Percent of final forgery attempts that were successful attacks
### Results: Estimate of Adversarial Success

**Forgery**: adversary is pretending to be a specific target author.

**Masking**: adversary is obscuring the original author.

<table>
<thead>
<tr>
<th></th>
<th>$C_5$</th>
<th>$C_{20}$</th>
<th>$C_{50}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Forgery</strong></td>
<td>66.6%</td>
<td>70.0%</td>
<td>73.0%</td>
</tr>
<tr>
<td><strong>Masking</strong></td>
<td>76.6%</td>
<td>76.6%</td>
<td>86.6%</td>
</tr>
</tbody>
</table>

Percent of final forgery attempts that were successful attacks
A **successful** forgery attack means the classifier output the target author instead of the original author of the code. 66.6% of forgery attacks against the $C_5$ classifier were successful.

<table>
<thead>
<tr>
<th></th>
<th>$C_5$</th>
<th>$C_{20}$</th>
<th>$C_{50}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Forgery</strong></td>
<td>66.6%</td>
<td>70.0%</td>
<td>73.0%</td>
</tr>
<tr>
<td><strong>Masking</strong></td>
<td>76.6%</td>
<td>76.6%</td>
<td>86.6%</td>
</tr>
</tbody>
</table>

Percent of final forgery attempts that were successful attacks
Results: Estimate of Adversarial Success

C50 attributed forgeries correctly only 13.4% of the time.

<table>
<thead>
<tr>
<th></th>
<th>$C_5$</th>
<th>$C_{20}$</th>
<th>$C_{50}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forgery</td>
<td>66.6%</td>
<td>70.0%</td>
<td>73.0%</td>
</tr>
<tr>
<td>Masking</td>
<td>76.6%</td>
<td>76.6%</td>
<td>86.6%</td>
</tr>
</tbody>
</table>

Percent of final forgery attempts that produced a misclassification
Lesson: Non-experts can successfully attack this state-of-the-art classifier, suggesting other authorship classifiers may be vulnerable to the same type of attacks.

<table>
<thead>
<tr>
<th></th>
<th>$C_5$</th>
<th>$C_{20}$</th>
<th>$C_{50}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Forgery</strong></td>
<td>66.6%</td>
<td>70.0%</td>
<td>73.0%</td>
</tr>
<tr>
<td><strong>Masking</strong></td>
<td>76.6%</td>
<td>76.6%</td>
<td>86.6%</td>
</tr>
</tbody>
</table>

Percent of final forgery attempts that produced a misclassification
Outline

- Motivation and Research Question
- Source Code Attribution: Overview and Background
- Evading Source Code Attribution: Definitions and Goals
- Methodology
- Results: Conservative Estimate of Adversarial Success
- Results: How to Create Forgeries
Lesson: Forgers did not know the features the classifier was using for attribution. This suggests that forgeries in the wild might contain the same types of modifications.
Example: Two Programs by Author C

// libraries imported
#define REP(i,a,b) for(i=a;i<b;i++)
#define rep(i,n) REP(i,0,n)
// variables defined
int main()
{
    int i, j, k, l, m, n, st;
    char in[10000];
    int fg[5000], chk[128];
    int size, count = 0, res;
    scanf ("%d%d%d", &len, &n, &size);
    rep (i, n) scanf ("%s", dic[i]);

    while (size--)
    {
        scanf ("%s", in);
        st = 0;
        rep (k, n) fg[k] = 1;

        while (size--)
        {
            scanf ("%d", &len);
            scanf ("%s", in);
            st = 0;
            rep (k, n) fg[k] = 1;
        }
    }
}
// libraries imported
#define REP(i,a,b) for(i=a;i<b;i++)
#define rep(i,n) REP(i,0,n)

// variables defined
int main() {
    int i, j, k, l, m, n, st;
    char in[10000];
    int fg[5000], chk[128];
    int size, count = 0, res;
    scanf ("%d%d%d", &len, &n, &size);
    rep(i, n) scanf("%s", dic[i]);

    while (size--){
        scanf("%s", in);
        st = 0;
        rep(k, n) fg[k] = 1;

        while (size--){
            scanf("%s", in);
            st = 0;
            rep(k, n) fg[k] = 1;
        }
    }

    Example: Two Programs by Author C

// libraries imported
#define REP(i,a,b) for(i=a;i<b;i++)
#define rep(i,n) REP(i,0,n)

// variables defined
int main() {
    int i, j, k, l, m, n, t, ok;
    int a, b, c;
    int size, count = 0;
    scanf("%d", &size);
    while (size--){
        scanf("%d%d", &n, &m);
        rep(i, m) {
            scanf("%d", s + i);
        }
    }

    Example: Two Programs by Author C
Example: Forgery of Author C

<table>
<thead>
<tr>
<th>Information Structure</th>
<th>Control Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Variable name</td>
<td>● Loop type</td>
</tr>
<tr>
<td>● Syntax</td>
<td></td>
</tr>
<tr>
<td>● Macros</td>
<td></td>
</tr>
<tr>
<td>● API calls</td>
<td></td>
</tr>
</tbody>
</table>
Example: Creating a Forgery of Author C

```cpp
int main()
{
    int i,j,k;
    int cc,ca;
    cin >> ca;
    for(cc=1;cc<=ca;cc++)
    {
        cin >> D >> I >> M >> N;
        for(i=0; i<N; i++)
            cin >> original[i];
    ...

    Classifier output: A
```
int main()
{
    int i,j,k;
    int cc,ca;
    cin >> ca;
    for (cc=1;cc<=ca;cc++)
    {
        cin >> D >> I >> M >> N;
        for(i=0; i<N; i++)
            cin >> original[i];
    ...  
}

Classifier output: A

Classifier output: ??
int main()
{
    int i,j,k;
    int cc,ca;
    cin >> ca;
    for(cc=1;cc<=ca;cc++)
    {
        cin >> D >> I >> M >> N;
        for(i=0; i<N; i++)
            cin >> original[i];
        ...
    }
...
```c
int main()
{
    int i,j,k;
    int cc,ca;
    cin >> ca;
    for(cc=1;cc<=ca;cc++)
    {
        cin >> D >> I >> M >> N;
        for(i=0; i<N; i++)
            cin >> original[i];
    }
    
    ...  
}
```

---

```c
int main()
{
    int i,j,k;
    int size, count = 0;
    cin >> size;
    for(count=1;count<=size;count++)
    {
        cin >> D >> I >> M >> N;
        for(i=0; i<N; i++)
            cin >> original[i];
    }

define REP(i,a,b) for(i=a;i<b;i++)
define rep(i,n) REP(i,0,n)
```

---

Classifier output: A

Classifier output: ??
#define REP(i,a,b) for(i=a;i<b;i++)
#define rep(i,n) REP(i,0,n)

int main()
{
    int i,j,k;
    int cc,ca;
    cin >> ca;
    for(cc=1; cc<=ca; cc++)
    {
        cin >> D >> I >> M >> N;
        for(i=0; i<N; i++)
            cin >> original[i];
    }
    ...

Classifier output: A

int main()
{
    int i,j,k;
    int size, count = 0;
    cin >> size;
    for(count=1; count<=size; count++)
    {
        cin >> D >> I >> M >> N;
        for(i=0; i<N; i++)
            cin >> original[i];
    }
    ...

Classifier output: ??
```
int main()
{
    int i,j,k;
    int cc,ca;
    cin >> ca;
    for(cc=1;cc<=ca;cc++)
    {
        cin >> D >> I >> M >> N;
        for(i=0; i<N; i++)
            cin >> original[i];
    }
...
```

```
#define REP(i,a,b) for(i=a;i<b;i++)
#define rep(i,n) REP(i,0,n)

int main()
{
    int i,j,k;
    int size, count = 0;
    scanf("%d", &size);
    for(count=1;count<=size;count++)
    {
        scanf("%d%d%d%d", &D, &I, &M, &N);
        for(i=0; i<N; i++)
            scanf("%d", original+i);
    }
...
```
int main()
{
    int i,j,k;
    int cc,ca;
    cin >> ca;
    for(cc=1;cc<=ca;cc++)
    {
        cin >> D >> I >> M >> N;
        for(i=0; i<N; i++)
            cin >> original[i];
    }
...

Classifier output: A

#define REP(i,a,b) for(i=a;i<b;i++)
#define rep(i,n) REP(i,0,n)

int main()
{
    int i,j,k;
    int size, count = 0;
    scanf("%d", &size);
    while (size--)
    {
        scanf("%d%d%d%d", &D, &I, &M, &N);
        rep (i,N)
            scanf("%d", original+i);
    }
...

Classifier output: ??
```cpp
int main()
{
    int i, j, k;
    int cc, ca;
    cin >> ca;
    for (cc=1; cc<=ca; cc++)
    {
        cin >> D >> I >> M >> N;
        for (i=0; i<N; i++)
            cin >> original[i];
    }
}
```

```cpp
int main()
{
    int i, j, k;
    int size, count = 0;
    scanf("%d", &size);
    while (size--)
    {
        scanf("%d%d%d%d", &D, &I, &M, &N);
        rep (i, N) scanf("%d", original+i);
    }
}
```
```c
int main()
{
    int i, j, k;
    int cc, ca;
    cin >> ca;
    for (cc=1; cc<=ca; cc++)
    {
        cin >> D >> I >> M >> N;
        for (i=0; i<N; i++)
            cin >> original[i];
    }
}
```

### FORGERY

```c
#define REP(i,a,b) for(i=a;i<b;i++)
#define rep(i,n) REP(i,0,n)

int main()
{
    int i, j, k;
    int size, count = 0;
    scanf("%d", &size);
    while (size--)
    {
        scanf("%d%d%d%d", &D, &I, &M, &N);
        rep(i,N) scanf("%d", original+i);
    }
}
```
Results: Methods of Forgery Creation

<table>
<thead>
<tr>
<th>Information Structure</th>
<th>Control Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Variable name</td>
<td>● Loop type</td>
</tr>
<tr>
<td>● Syntax</td>
<td></td>
</tr>
<tr>
<td>● Macros</td>
<td></td>
</tr>
<tr>
<td>● API calls</td>
<td></td>
</tr>
</tbody>
</table>
## Results: Methods of Forgery Creation

<table>
<thead>
<tr>
<th>Information Structure</th>
<th>Control Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Variable name</td>
<td>● Loop type</td>
</tr>
<tr>
<td>● Syntax</td>
<td>● If-statements</td>
</tr>
<tr>
<td>● Macros</td>
<td>● Assignments per line</td>
</tr>
<tr>
<td>● API calls</td>
<td>● Control flow keywords</td>
</tr>
<tr>
<td>● Libraries imported</td>
<td>● Loop logic</td>
</tr>
<tr>
<td>● Variable decl location</td>
<td></td>
</tr>
</tbody>
</table>
**Results: Methods of Forgery Creation**

<table>
<thead>
<tr>
<th>Information Structure</th>
<th>Control Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Variable name</td>
<td>● Loop type</td>
</tr>
<tr>
<td>● Syntax</td>
<td>● If-statements</td>
</tr>
<tr>
<td>● Macros</td>
<td>● Assignments per line</td>
</tr>
<tr>
<td>● API calls</td>
<td>● Control flow keywords</td>
</tr>
<tr>
<td>● Libraries imported</td>
<td>● Loop logic</td>
</tr>
<tr>
<td>● Variable decl location</td>
<td></td>
</tr>
</tbody>
</table>

**Local modifications:** only need to understand a line or two of code
Results: Methods of Forgery Creation

<table>
<thead>
<tr>
<th>Information Structure</th>
<th>Control Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Variable name</td>
<td>● Loop type</td>
</tr>
<tr>
<td>● Syntax</td>
<td>● If-statements</td>
</tr>
<tr>
<td>● Macros</td>
<td>● Assignments per line</td>
</tr>
<tr>
<td>● API calls</td>
<td>● Control flow keywords</td>
</tr>
<tr>
<td>● Libraries imported</td>
<td>● Loop logic</td>
</tr>
<tr>
<td>● Variable decl location</td>
<td></td>
</tr>
</tbody>
</table>

**Local modifications**: only need to understand a line or two of code

**Algorithmic modifications**: need a more comprehensive understanding of the code
# Results: Methods of Forgery Creation

<table>
<thead>
<tr>
<th>Information Structure</th>
<th>Control Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Variable name</td>
<td>● Loop type</td>
</tr>
<tr>
<td>● Syntax</td>
<td>● If-statements</td>
</tr>
<tr>
<td>● Macros</td>
<td>● Assignments per line</td>
</tr>
<tr>
<td>● API calls</td>
<td>● Control flow keywords</td>
</tr>
<tr>
<td>● Libraries imported</td>
<td>● Loop logic</td>
</tr>
<tr>
<td>● Variable decl location</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Local modifications**: only need to understand a line or two of code

**Algorithmic modifications**: need a more comprehensive understanding of the code

<table>
<thead>
<tr>
<th>Local</th>
<th>Algorithmic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Results: Methods of Forgery Creation

<table>
<thead>
<tr>
<th>Information Structure</th>
<th>Control Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Variable name</td>
<td>● Loop type</td>
</tr>
<tr>
<td>● Syntax</td>
<td>● If-statements</td>
</tr>
<tr>
<td>● Macros</td>
<td>● Assignments per line</td>
</tr>
<tr>
<td>● API calls</td>
<td>● Control flow keywords</td>
</tr>
<tr>
<td>● Libraries imported</td>
<td>● Loop logic</td>
</tr>
<tr>
<td>● Variable decl location</td>
<td></td>
</tr>
</tbody>
</table>

### Local modifications:
- only need to understand a line or two of code

### Algorithmic modifications:
- need a more comprehensive understanding of the code

Local Algorithmic

X

Local

Algorithmic

X
## Results: Methods of Forgery Creation

<table>
<thead>
<tr>
<th>Information Structure</th>
<th>Control Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Variable name</td>
<td>● Loop type</td>
</tr>
<tr>
<td>● Syntax</td>
<td>● If-statements</td>
</tr>
<tr>
<td>● Macros</td>
<td>● Assignments per line</td>
</tr>
<tr>
<td>● API calls</td>
<td>● Control flow keywords</td>
</tr>
<tr>
<td>● Libraries imported</td>
<td>● Loop logic</td>
</tr>
<tr>
<td>● Variable decl location</td>
<td></td>
</tr>
</tbody>
</table>

### Local modifications:
- only need to understand a line or two of code

### Algorithmic modifications:
- need a more comprehensive understanding of the code
Lessons from methods of forgery creation:

- Local modifications are common.
- Some forgers copied code directly the target author’s training set.
Programmers desiring privacy or with malicious intent may seek to evade source code attribution classifiers.

Lab study with C programmers producing forgeries, showing unsophisticated adversaries can fool a state of the art classifier.

Forgeries were successful with local changes that do not require a high-level understanding of the programming style.

More recommendations in paper!

My coauthors: Luke Zettlemoyer, Tadayoshi Kohno
Contact me: Lucy Simko, simkol@cs.washington.edu, https://homes.cs.washington.edu/~simkol/