P2C:
Understanding Output Data Files via On-the-Fly Transformation from Producer to Consumer Execution

Yonghwi Kwon, Fei Peng, Dohyeong Kim, Kyungtae Kim, Xiangyu Zhang, Dongyan Xu
Department of Computer Science, Purdue University

Vinod Yegneswaran
SRI International

John Qian
Cisco Systems
Motivation

• *Understanding unknown data files and network messages is a prominent security challenge*
Motivation

- Without audit logging/monitoring systems
  - Who created the files?
  - Do they contain private data?
    - Personal profile
    - Contact list
    - Key-strokes

- Found mysterious binary files
Motivation

- **Existing solutions require consumer programs**
  - Input format reverse engineering based on consumers
    - Prospex, Tupni, REWARDS, Dispatcher, Reformat, ...
  - Monitoring the execution of consumer to analyze how the files/messages are parsed

```
  fread(&buf, ..., 1, f);
  ...
  strcpy( name, buf );
  ...
```

---

Field’s type is a string

Reads a field from a file
Motivation

• What if we only have the producer program?
  • Botnet Command and Control (C&C) protocol
    ▪ No consumer is present on the victim machine
    ▪ Consumer exists on the attacker’s server
Motivation

- What if we only have the producer program?
  - Botnet Command and Control (C&C) protocol
    - No consumer is present on the victim machine
    - Consumer exists on the attacker’s server

![Diagram showing producer and consumer](image-url)
Observation

• Producer and consumer are symmetric
  • “Checking Conformance of a Producer and a Consumer”, Evan Driscoll, Amanda Burton, and Thomas Reps, FSE’11
  • The correctness of a producer can be verified by checking its conformance to the corresponding consumer
Observation

- Producer and consumer are symmetric
- Consumer and Producer follow the same rule
  - `((boolean, double, boolean) | (boolean))`

### Producer

1. `sendReading(Sensor* device, int prev)`
2. `if device→setting == prev then`
   - `writeBool(false);`
3. `else`
4. `writeBool(true);`
5. `writeDouble(device→setting);`
6. `writeBool(device→valid);`

### Consumer

1. `updateReading(int* setting, bool* valid)`
2. `*setting = readDouble();`
3. `*valid = readBool();`

### Main

```c
int sehng;
bool valid;
while ...
  if readBool() then
    updateReading(&setting, &valid);
  ...
```

- // do something with current readings
Observation

- Producer and consumer are symmetric
- Consumer and Producer follow the same rule
  - ((boolean, double, boolean) | (boolean))*

Our idea:
Run producer to create consumer (P2C)
Forced execution

- Executing producer w/o proper inputs
  - May do different tasks and quit, without creating a file
Forced execution

• Our earlier tool comes to the rescue
• Our forced execution engine: X-force

- Generating random inputs
- Recovering from faults
Overview

• Basic idea
  • Given an unknown file/message and a potential producer
Overview

• Basic idea
  • Given an **unknown file/message** and a **potential producer**
  • Explore execution paths that contain “file open-for-write” operations (Producer executions)
    ▪ Leveraging a binary forced execution technique (X-force)
Overview

• Basic idea
  • Given an unknown file/message and a potential producer
  • Explore execution paths that contain “file open-for-write” operations (Producer executions)
    ▪ Leveraging a binary forced execution technique (X-force)
  • Transform the producer to consumer execution
Overview

• Basic idea
  • Given an **unknown file/message** and a **potential producer**
  • Explore execution paths that contain “file open-for-write” operations (Producer executions)

  § Leveraging a binary forced execution technique (X-force)

• Transform the producer to consumer execution

P2C directly works on potential producer programs

P2C does not require
  (1) consumers
  (2) inputs to producers
An unknown binary file

| 0x4bfd | 0x5 | 0x15 | 0x2 | 0x2 | “silver_member” | “john12” | 2001 | ... |
How it is generated?

- **Unknown file**

```
0x4bfd | 0x5 | 0x15 | 0x2 | 0x2 | “silver_member” | “john12” | 2001 | ...
```

**Producer**

```c
1    f = fopen( ..., “w” );
2    account_num = ...;
3    int magic = 0x4bfd;
4    fwrite(&magic, sizeof(int), 1, f);
5    fwrite(&account_num, sizeof(int), 1, f);
6    header->name = ...;
7    header->acnt_number = ...;
8    size = ...;
9    fwrite(&size, sizeof(int), 1, f);
10   fwrite(header, size, 1, f);
11   for( i = 0; i < account_num; i++ )
12   fwrite(account[0], ..., f);
```

// Data Structures
struct tag_header {
    int type;
    int acnt_num;
    char name[1];
} *header;

struct tag_account_entry {
    char id[32];
    int year;
    int month;
    int day;
    int balance;
} *account;
```
Producer w/o proper inputs (X-force)

- **Unknown file**

| 0x4bfd | 0x5 | 0x15 | 0x2 | “silver_member” | “john12” | 2001 | ... |

```c
Producer
1  f = fopen( ..., “w”);
2  account_num = ...; // 2
3  int magic = 0x4bfd;
4  fwrite(&magic, sizeof(int), 1, f);
5  fwrite(&account_num, sizeof(int), 1, f);
6  header->name = ...; // “#” 
7  header->acnt_number = ...; // 2
8  size = ...
9  fwrite(&size, sizeof(int), 1, f);
10 fwrite(header, size, 1, f);
11 for( i = 0; i < account_num; i++)
12  fwrite(account[0], ..., f);
```

// Data Structures
```c
struct tag_header {
  int type;
  int acnt_num;
  char name[1];
} *header;
```

```c
struct tag_account_entry {
  char id[32];
  int year;
  int month;
  int day;
  int balance;
} *account;
```
Producer w/o proper inputs (X-force)

- **Unknown file**

<table>
<thead>
<tr>
<th>Producer</th>
<th>header</th>
<th>account</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4bfd</td>
<td>0x5</td>
<td>0x15</td>
</tr>
<tr>
<td></td>
<td>0x2</td>
<td>0x2</td>
</tr>
<tr>
<td>“silver_member”</td>
<td>“john12”</td>
<td>2001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

- **File generated by the forced execution**

<table>
<thead>
<tr>
<th>Producer</th>
<th>header</th>
<th>account</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4bfd</td>
<td>0x2</td>
<td>0xA</td>
</tr>
<tr>
<td></td>
<td>0x2</td>
<td>0x2</td>
</tr>
<tr>
<td>“#”</td>
<td>“#_+”</td>
<td>2001</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

```c
Producer
1  f = fopen(..., "w" );
    ...
2  account_num = ...; // 2
    ...
3  int magic = 0x4bfd;
4  fwrite(&magic, sizeof(int), 1, f);
5  fwrite(&account_num, sizeof(int), 1, f);
6  header->name = ...; // "#_
7  header->acnt_number = ...; // 2
8  size = ...;
9  fwrite(&size, sizeof(int), 1, f);
10 fwrite(header, size, 1, f);
11 for( i = 0; i < account_num; i++ )
12    fwrite(account[0], ..., f);
```

// Data Structures
struct tag_header {
  int type;
  int acnt_num;
  char name[1];
} *header;

struct tag_account_entry {
  char id[32];
  int year;
  int month;
  int day;
  int balance;
} *account;

```c```
Producer w/o proper inputs (X-force)

- **Unknown file**
  - Producer
    ```
    1 f = fopen( ..., "w" );
    2 account_num = ...; // 2
    3 int magic = 0x4bfd;
    4 fwrite(&magic, sizeof(int), 1, f);
    5 fwrite(&account_num, sizeof(int), 1, f);
    6 header->name = ...; // ".
    7 header->acnt_number = ...; // 2
    8 size = ...;
    9 fwrite(&size, sizeof(int), 1, f);
    10 fwrite(header, size, 1, f);
    11 for( i = 0; i < account_num; i++ )
    12 fwrite(account[0], ..., f);
    ```
  - // Data Structures
    ```
    struct tag_header {
        int type;
        int acnt_num;
        char name[1];
    } *header;
    ```
  - struct tag_account_entry {
    ```
    char id[32];
    int year;
    int month;
    int day;
    int balance;
    ```
  - } *account;

- **File generated by the forced execution**
  - Size of header
  - 0x4bfd 0x5 0x15 0x2 0x2 “silver_member” “john12” 2001 ...
  - header
  - 0x4bfd 0x2 0xA 0x2 0x2 “#” “#_” 2001 ...
  - account
### Producer w/o proper inputs (X-force)

1. **Unknown file**
   
   ![Diagram of unknown file structure](image)

   ```
   Producer
   1   f = fopen( ..., “w” );
   2   account_num = ...; // 2
   3   int magic = 0x4bfd;
   4   fwrite(&magic, sizeof(int), 1, f);
   5   fwrite(&account_num, sizeof(int), 1, f);
   6   header->name = ...; // “#_”
   7   header->acnt_number = ...; // 2
   8   size = ...;
   9   fwrite(&size, sizeof(int), 1, f);
   10  fwrite(header, size, 1, f);
   11  for( i = 0; i < account_num; i++ )
   12  fwrite(account[0], ..., f);
   ```

2. **File generated by the forced execution**
   
   ![Diagram of file structure](image)

   ```
   // Data Structures
   struct tag_header {
     int type;
     int acnt_num;
     char name[1];
   } *header;
   ```

   ```
   struct tag_account_entry {
     char id[32];
     int year;
     int month;
     int day;
     int balance;
   } *account;
   ```
Transforming a producer to a consumer

- **Unknown file**
  
  | 0x4bfd | 0x5 | 0x15 | 0x2 | 0x2 | “silver_member” | “john12” | 2001 | ... |
  
- **File generated by the forced execution**
  
  | 0x4bfd | 0x2 | 0xA | 0x2 | 0x2 | “#_” | “#_+” | 2001 | ... |

### Producer

```c
1  
   f = fopen( ..., “w” );

   ... 

2  account_num = ...; // 2

   ...

3  int magic = 0x4bfd;

4  fwrite(&magic, sizeof(int), 1, f);

5  fwrite(&account_num, sizeof(int), 1, f);

6  header->name = ...; // “#_”

7  header->acnt_number = ...; // 2

8  size = ...;

9  fwrite(&size, sizeof(int), 1, f);

10 fwrite(header, size, 1, f);

11 for( i = 0; i < account_num; i++ )
12   fwrite(account[0], ..., f);
```

### Consumer

```c
1  
   f = fopen( ..., “r” );

   ... 

2  account_num = ...; // 2

   ...

3  int magic = 0x4bfd;

4  fread(&magic, sizeof(int), 1, f);

5  fread(&account_num, sizeof(int), 1, f);

6  header->name = ...; // “#_”

7  header->acnt_number = ...; // 2

8  size = ...;

9  fread(&size, sizeof(int), 1, f);

10 fread(header, size, 1, f);

11 for( i = 0; i < account_num; i++ )
12   fread(account[0], ..., f);
```
Transforming a producer to a consumer

- **Unknown file**
  
  | 0x4bfd | 0x5 | 0x15 | 0x2 | 0x2 | “silver_member” | “john12” | 2001 | ...

- **File generated by the forced execution**
  
  | 0x4bfd | 0x2 | 0xA | 0x2 | 0x2 | “#_” | “#_+” | 2001 | ...

**Producer**

```c
1   f = fopen( ..., “w” );
    ...
2   account_num = ...; // 2
    ...
3   int magic = 0x4bfd;
4   fwrite(&magic, sizeof(int), 1, f);
5   fwrite(&account_num, sizeof(int), 1, f);
6   header->name = ...; // “#_”
7   header->acnt_number = ...; // 2
8   size = ...;
9   fwrite(&size, sizeof(int), 1, f);
10  fwrite(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12   fwrite(account[0], ..., f);
```

**Consumer**

```c
1   f = fopen( ..., “r” );
    ...
2   account_num = ...; // 2
    ...
3   int magic = 0x4bfd;
4   fread(&magic, sizeof(int), 1, f);
5   fread(&account_num, sizeof(int), 1, f);
6   header->name = ...; // “#_”
7   header->acnt_number = ...; // 2
8   size = ...;
9   fread(&size, sizeof(int), 1, f);
10  fread(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12   fread(account[0], ..., f);
```
Transforming a producer to a consumer

- **Unknown file**

  ```
  0x4bfd  0x5   0x15  0x2   0x2  "silver_member"  "john12"  2001  ...
  ```

- **File generated by the forced execution**

  ```
  0x4bfd  0x5   0xA  0x2   0x2  "#_"  "#_+"  2001  ...
  ```

**Producer**

1. `f = fopen( ..., "w" );`
   ...
2. `account_num = ...; // 2`
   ...
3. `int magic = 0x4bfd;`
4. `fwrite(&magic, sizeof(int), 1, f);`
5. `fwrite(&account_num, sizeof(int), 1, f);`
6. `header->name = ...; // "#_"
7. `header->acnt_number = ...; // 2`
8. `size = ...;
9. `fwrite(&size, sizeof(int), 1, f);`
10. `fwrite(header, size, 1, f);`
11. `for( i = 0; i < account_num; i++ )`
12. `fwrite(account[0], ..., f);`

**Consumer**

1. `f = fopen( ..., "r" );`
   ...
2. `account_num = ...; // 2`
   ...
3. `int magic = 0x4bfd;`
4. `fread(&magic, sizeof(int), 1, f);`
5. `fread(&account_num, sizeof(int), 1, f);`
6. `header->name = ...; // "#_"
7. `header->acnt_number = ...; // 2`
8. `size = ...;
9. `fread(&size, sizeof(int), 1, f);`
10. `fwrite(header, size, 1, f);`
11. `for( i = 0; i < account_num; i++ )`
12. `fwrite(account[0], ..., f);`
Transforming a producer to a consumer

• Unknown file

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>“silver_member”</th>
<th>“john12”</th>
<th>2001</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4bfd</td>
<td>0x5</td>
<td>0x15</td>
<td>0x2</td>
<td>0x2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• File generated by the forced execution

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>“#_”</th>
<th>“#_+”</th>
<th>2001</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4bfd</td>
<td>0x5</td>
<td>0x15</td>
<td>0x2</td>
<td>0x2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Producer**

1. `f = fopen( ..., “w” );`
2. `account_num = ...; // 2`
3. `int magic = 0x4bfd;`
4. `fwrite(&magic, sizeof(int), 1, f);`
5. `fwrite(&account_num, sizeof(int), 1, f);`
6. `header->name = ...; // “#_”`
7. `header->acnt_number = ...; // 2`
8. `size = ...;`
9. `fwrite(&size, sizeof(int), 1, f);`
10. `fwrite(header, size, 1, f);`
11. `for( i = 0; i < account_num; i++ )`
12. `fwrite(account[0], ..., f);`

**Consumer**

1. `f = fopen( ..., “r” );`
2. `account_num = ...; // 2`
3. `int magic = 0x4bfd;`
4. `fread(&magic, sizeof(int), 1, f);`
5. `fread(&account_num, sizeof(int), 1, f);`
6. `header->name = ...; // “#_”`
7. `header->acnt_number = ...; // 2`
8. `size = ...;`
9. `fread(&size, sizeof(int), 1, f);`
10. `fread(header, size, 1, f);`
11. `for( i = 0; i < account_num; i++ )`
12. `fread(account[0], ..., f);`
Transforming a producer to a consumer

• **Unknown file**

```
0x4bfd  0x5  0x15  0x2  0x2  "silver_member"  "john12"  2001  ...
```

• **File generated by the forced execution**

```
0x4bfd  0x5  0x15  0x2  0x2  "silver_member"  "#_+"  2001  ...
```

**Producer**

```
1   f = fopen( ..., "w" );
    ...
2   account_num = ...; // 2  
    ...
3   int magic = 0x4bfd;
4   fwrite(&magic, sizeof(int), 1, f);
5   fwrite(&account_num, sizeof(int), 1, f);  
6   header->name = ...; // "#_
7   header->acnt_number = ...; // 2  
8   size = ...;
9   fwrite(&size, sizeof(int), 1, f);
10  fwrite(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12   fwrite(account[0], ..., f);
```

**Consumer**

```
1   f = fopen( ..., "r" );
    ...
2   account_num = ...; // 2  
    ...
3   int magic = 0x4bfd;
4   fread(&magic, sizeof(int), 1, f);
5   fread(&account_num, sizeof(int), 1, f);  
6   header->name = ...; // "#_
7   header->acnt_number = ...; // 2  
8   size = ...;
9   fread(&size, sizeof(int), 1, f);
10  fread(header, size, 1, f);
11  for( i = 0; i < account_num; i++ )
12   fread(account[0], ..., f);
```
Transforming a producer to a consumer

- **Unknown file**

<table>
<thead>
<tr>
<th>magic</th>
<th>account_num</th>
<th>header-&gt;name</th>
<th>header-&gt;acnt_number</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4bfd</td>
<td>0x5</td>
<td>0x15</td>
<td>0x2</td>
<td>0x2</td>
</tr>
<tr>
<td>“silver_member”</td>
<td>“john12”</td>
<td>2001</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

- **File generated by the forced execution**

<table>
<thead>
<tr>
<th>magic</th>
<th>account_num</th>
<th>header-&gt;name</th>
<th>header-&gt;acnt_number</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4bfd</td>
<td>0x5</td>
<td>0x15</td>
<td>0x2</td>
<td>0x2</td>
</tr>
<tr>
<td>“silver_member”</td>
<td>“john12”</td>
<td>2001</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

**Producer**

```c
1  f = fopen( ..., “w” );
   ...
2  account_num = ...; // 2
   ...
3  int magic = 0x4bfd;
4  fwrite(&magic, sizeof(int), 1, f);
5  fwrite(&account_num, sizeof(int), 1, f);
6  header->name = ...; // “#_”
7  header->acnt_number = ...; // 2
8  size = ...;
9  fwrite(&size, sizeof(int), 1, f);
10 fwrite(header, size, 1, f);
11 for( i = 0; i < account_num; i++ )
12    fwrite(account[0], ..., f);
```

**Consumer**

```c
1  f = fopen( ..., “r” );
   ...
2  account_num = ...; // 2
   ...
3  int magic = 0x4bfd;
4  fread(&magic, sizeof(int), 1, f);
5  fread(&account_num, sizeof(int), 1, f);
6  header->name = ...; // “#_”
7  header->acnt_number = ...; // 2
8  size = ...;
9  fread(&size, sizeof(int), 1, f);
10 fread(header, size, 1, f);
11 for( i = 0; i < account_num; i++ )
12    fread(account[0], ..., f);
```
Technical Challenges

• **Unexposed field correlations**
  • *Symmetric read operations do not help*
  • *Semantically related read operations do not have explicit dependence*

```
Producer
1  f = fopen( ... );
2  ...
3  len = strlen(buf);
4  ...
5  fwrite(&len, sizeof(int), 1, f);
6  fwrite(buf, strlen(buf), 1, f);

Consumer
1  f = fopen( ... );
2  ...
3  len = strlen(buf);
4  ...
5  fread(&len, sizeof(int), 1, f); // (1)
6  fread(buf, strlen(buf), 1, f); // (2)
7  // no dependence path between (1) and (2)
```
Technical Challenges

• **Unexposed field correlations**
  
  • **Original Definition Tracking**
    - Change the original definition of variables
  
  • **Patching and Tracing**
    - Iteratively run the program for consistency checking
    - Inconsistency indicates the presence of unexposed field correlations
    - Try a different value
      (e.g. try to read more bytes)
Evaluation

- **5 programs generate files / 4 programs generate network messages**

<table>
<thead>
<tr>
<th>Program</th>
<th>Program Size</th>
<th>Unknown file size</th>
<th># of iteration (Matched / Unmatched)</th>
<th># of typed fields / # of fields</th>
<th>Elapsed Time</th>
<th>Coverage / # of total inst.</th>
<th># of paths explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfoZip</td>
<td>37KB</td>
<td>5KB</td>
<td>12 / 0</td>
<td>19 / 35</td>
<td>2m 45s</td>
<td>6015/6015</td>
<td>385</td>
</tr>
<tr>
<td>Steganography</td>
<td>17KB</td>
<td>4219KB</td>
<td>24 / 0</td>
<td>3 / 3</td>
<td>9m 27s</td>
<td>631/974</td>
<td>12</td>
</tr>
<tr>
<td>FreePiano</td>
<td>2296KB</td>
<td>6KB</td>
<td>22 / 0</td>
<td>6 / 6</td>
<td>28m 35s</td>
<td>47991/286571</td>
<td>2531</td>
</tr>
<tr>
<td>Mp3gain</td>
<td>109KB</td>
<td>1304KB</td>
<td>17 / 0</td>
<td>6 / 10</td>
<td>24m 42s</td>
<td>21672/21672</td>
<td>754</td>
</tr>
<tr>
<td>Yamdi</td>
<td>225KB</td>
<td>102KB</td>
<td>38 / 0</td>
<td>34 / 72</td>
<td>1h 16m 38s</td>
<td>21491/24577</td>
<td>1183</td>
</tr>
<tr>
<td>Zbot</td>
<td>26KB</td>
<td>30Bytes</td>
<td>7 / 2</td>
<td>5 / 7</td>
<td>8m 48s</td>
<td>3089/3089</td>
<td>199</td>
</tr>
<tr>
<td>Winping</td>
<td>244KB</td>
<td>45Bytes</td>
<td>5 / 0</td>
<td>4 / 8</td>
<td>2m 27s</td>
<td>14758/29620</td>
<td>983</td>
</tr>
<tr>
<td>PowerOn</td>
<td>25KB</td>
<td>102Bytes</td>
<td>17 / 0</td>
<td>2 / 2</td>
<td>709/1058</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>NetworkMorris</td>
<td>1049KB</td>
<td>52Bytes</td>
<td>20 / 4</td>
<td>15 / 39</td>
<td>8m 6s</td>
<td>3057/7814</td>
<td>186</td>
</tr>
</tbody>
</table>

5 programs generate files

4 programs generate network messages
## Evaluation

- **Matched:** When the file is correctly parsed
- **Unmatched:** When it fails to parse the file

<table>
<thead>
<tr>
<th>Program</th>
<th>Program Size</th>
<th>Unknown file size</th>
<th># of iteration (Matched / Unmatched)</th>
<th># of typed fields / # of fields</th>
<th>Elapsed Time</th>
<th>Coverage / # of total inst.</th>
<th># of paths explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfoZip</td>
<td>37KB</td>
<td>5KB</td>
<td>12 / 0</td>
<td>19 / 35</td>
<td>2m 45s</td>
<td>6015/6015</td>
<td>385</td>
</tr>
<tr>
<td>Steganography</td>
<td>17KB</td>
<td>4219KB</td>
<td>24 / 0</td>
<td>3 / 3</td>
<td>9m 27s</td>
<td>631/974</td>
<td>12</td>
</tr>
<tr>
<td>FreePiano</td>
<td>2296KB</td>
<td>6KB</td>
<td>42 / 0</td>
<td>6 / 6</td>
<td>28m 35s</td>
<td>47991/286571</td>
<td>2531</td>
</tr>
<tr>
<td>Mp3gain</td>
<td>109KB</td>
<td>1304KB</td>
<td>17 / 0</td>
<td>6 / 10</td>
<td>24m 42s</td>
<td>21672/21672</td>
<td>754</td>
</tr>
<tr>
<td>Yamdi</td>
<td>225KB</td>
<td>102KB</td>
<td>38 / 0</td>
<td>34 / 72</td>
<td>1h 16m 38s</td>
<td>21491/24577</td>
<td>1183</td>
</tr>
<tr>
<td>Zbot</td>
<td>26KB</td>
<td>30Bytes</td>
<td>7 / 2</td>
<td>5 / 7</td>
<td>8m 48s</td>
<td>3089/3089</td>
<td>199</td>
</tr>
<tr>
<td>Winping</td>
<td>244KB</td>
<td>45Bytes</td>
<td>5 / 0</td>
<td>4 / 8</td>
<td>2m 27s</td>
<td>14758/29620</td>
<td>983</td>
</tr>
<tr>
<td>PowerOn</td>
<td>25KB</td>
<td>102Bytes</td>
<td>1 / 0</td>
<td>2 / 2</td>
<td>21s</td>
<td>709/1058</td>
<td>29</td>
</tr>
<tr>
<td>NetworkMorris</td>
<td>1049KB</td>
<td>52Bytes</td>
<td>20 / 4</td>
<td>15 / 39</td>
<td>8m 6s</td>
<td>3057/7814</td>
<td>186</td>
</tr>
</tbody>
</table>
Evaluation

- # of typed fields / total # of fields

<table>
<thead>
<tr>
<th>Program</th>
<th>Program Size</th>
<th>Unknown file size</th>
<th># of iteration (Matched / Unmatched)</th>
<th># of typed fields / # of fields</th>
<th>Elapsed Time</th>
<th>Coverage / # of total inst.</th>
<th># of paths explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfoZip</td>
<td>37KB</td>
<td>5KB</td>
<td>12 / 0</td>
<td>19 / 35</td>
<td>2m 45s</td>
<td>6015/6015</td>
<td>385</td>
</tr>
<tr>
<td>Steganography</td>
<td>17KB</td>
<td>4219KB</td>
<td>24 / 0</td>
<td>3 / 3</td>
<td>9m 27s</td>
<td>631/974</td>
<td>12</td>
</tr>
<tr>
<td>FreePiano</td>
<td>2296KB</td>
<td>6KB</td>
<td>42 / 0</td>
<td>6 / 6</td>
<td>28m 35s</td>
<td>47991/286571</td>
<td>2531</td>
</tr>
<tr>
<td>Mp3gain</td>
<td>109KB</td>
<td>1304KB</td>
<td>17 / 0</td>
<td>6 / 10</td>
<td>24m 42s</td>
<td>21672/21672</td>
<td>754</td>
</tr>
<tr>
<td>Yamdi</td>
<td>225KB</td>
<td>102KB</td>
<td>38 / 0</td>
<td>34 / 72</td>
<td>1h 16m 38s</td>
<td>21491/24577</td>
<td>1183</td>
</tr>
<tr>
<td>Zbot</td>
<td>26KB</td>
<td>30Bytes</td>
<td>7 / 2</td>
<td>5 / 7</td>
<td>8m 48s</td>
<td>3089/3089</td>
<td>199</td>
</tr>
<tr>
<td>Winping</td>
<td>244KB</td>
<td>45Bytes</td>
<td>5 / 0</td>
<td>4 / 8</td>
<td>2m 27s</td>
<td>14758/29620</td>
<td>983</td>
</tr>
<tr>
<td>PowerOn</td>
<td>25KB</td>
<td>102Bytes</td>
<td>1 / 0</td>
<td>2 / 2</td>
<td>21s</td>
<td>709/1058</td>
<td>29</td>
</tr>
<tr>
<td>NetworkMorris</td>
<td>1049KB</td>
<td>52Bytes</td>
<td>20 / 4</td>
<td>15 / 39</td>
<td>8m 6s</td>
<td>3057/7814</td>
<td>186</td>
</tr>
</tbody>
</table>
Evaluation

- **Elapsed time (P2C alone):** 21s ~ 1h 16m

<table>
<thead>
<tr>
<th>Program</th>
<th>Program Size</th>
<th>Unknown file size</th>
<th># of iteration (Matched / Unmatched)</th>
<th># of typed fields/ # of fields</th>
<th>Elapsed Time</th>
<th>Coverage / # of total inst.</th>
<th># of paths explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfoZip</td>
<td>37KB</td>
<td>5KB</td>
<td>12 / 0</td>
<td>19 / 35</td>
<td>2m 45s</td>
<td>6015/6015</td>
<td>385</td>
</tr>
<tr>
<td>Steganography</td>
<td>17KB</td>
<td>4219KB</td>
<td>24 / 0</td>
<td>3 / 3</td>
<td>9m 27s</td>
<td>631/974</td>
<td>12</td>
</tr>
<tr>
<td>FreePiano</td>
<td>2296KB</td>
<td>6KB</td>
<td>42 / 0</td>
<td>6 / 6</td>
<td>28m 35s</td>
<td>47991/286571</td>
<td>2531</td>
</tr>
<tr>
<td>Mp3gain</td>
<td>109KB</td>
<td>1304KB</td>
<td>17 / 0</td>
<td>6 / 10</td>
<td>24m 42s</td>
<td>21672/21672</td>
<td>754</td>
</tr>
<tr>
<td>Yamdi</td>
<td>225KB</td>
<td>102KB</td>
<td>38 / 0</td>
<td>34 / 72</td>
<td>1h 16m 38s</td>
<td>21491/24577</td>
<td>1183</td>
</tr>
<tr>
<td>Zbot</td>
<td>26KB</td>
<td>30Bytes</td>
<td>7 / 2</td>
<td>5 / 7</td>
<td>8m 48s</td>
<td>3089/3089</td>
<td>199</td>
</tr>
<tr>
<td>Winping</td>
<td>244KB</td>
<td>45Bytes</td>
<td>5 / 0</td>
<td>4 / 8</td>
<td>2m 27s</td>
<td>14758/29620</td>
<td>983</td>
</tr>
<tr>
<td>PowerOn</td>
<td>25KB</td>
<td>102Bytes</td>
<td>1 / 0</td>
<td>2 / 2</td>
<td>21s</td>
<td>709/1058</td>
<td>29</td>
</tr>
<tr>
<td>NetworkMorris</td>
<td>1049KB</td>
<td>52Bytes</td>
<td>20 / 4</td>
<td>15 / 39</td>
<td>8m 6s</td>
<td>3057/7814</td>
<td>186</td>
</tr>
</tbody>
</table>
Case study

- **Steganography**
  - Embeds a secret text by changing the LSB (Least Significant Bit)s

Is there a hidden message or not?
Evaluation

- **Steganography**
  - Apply P2C to transform producer to consumer
  - Use existing taint analysis techniques to understand the secret message format
Case study

- Steganography

“LAST OF NDSS”

Is there a hidden message or not?
Case study

- Steganography
- Benign

Failed to transform the producer

Is there a hidden message or not?
Evaluation

• **Observations**
  • # of iterations to find the correct transformation is not large
    ▪ In most cases, program dependences are exposed.

• Can precisely identify all the fields in the given files/message
  ▪ Yamdi: Floating Point
  ▪ NetworkMorris/WinPing/InfoZip: fields are not related to any standard API

• Transformed consumer execution recognizes more fields than some typical consumers (mp3tag)
Conclusion

• **P2C: an output format reverse-engineering tool**
  • Understand the structure and meaning of unknown file/message
  • Based only on producer, without consumer

• **Key Idea**
  • Transforming a producer execution to the ideal consumer execution

• **Key Features**
  • Highly accurate
  • No need to know exact producer
  • No need to know how to run it
Question?

Thank you!

Email: yongkwon@purdue.edu