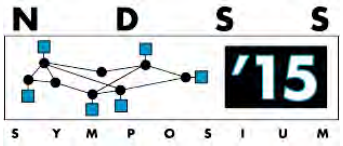
P2C:

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***

Downloads and Installs
few freeware programs

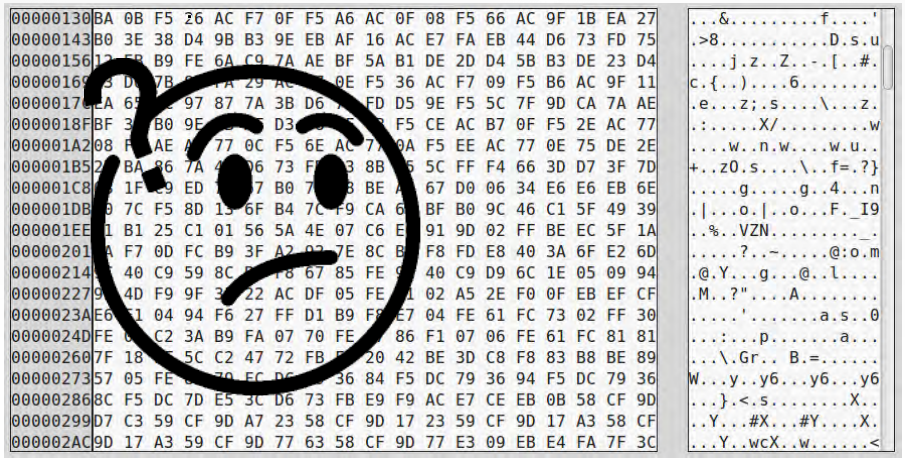
Weeks later

Found mysterious
binary files

Motivation

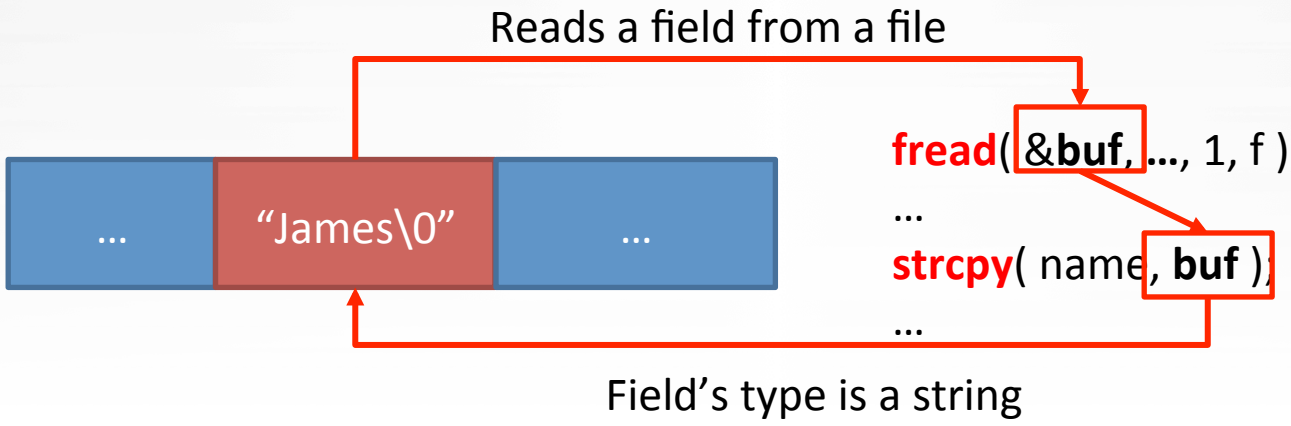
- ***Without audit logging/monitoring systems***
  - Who created the files?
  - Do they contain private data?
    - Personal profile
    - Contact list
    - Key-strokes

Found mysterious binary files



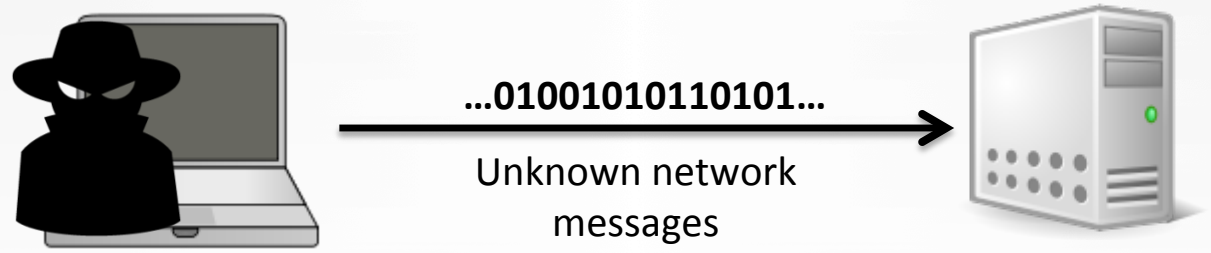
Motivation

- ***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



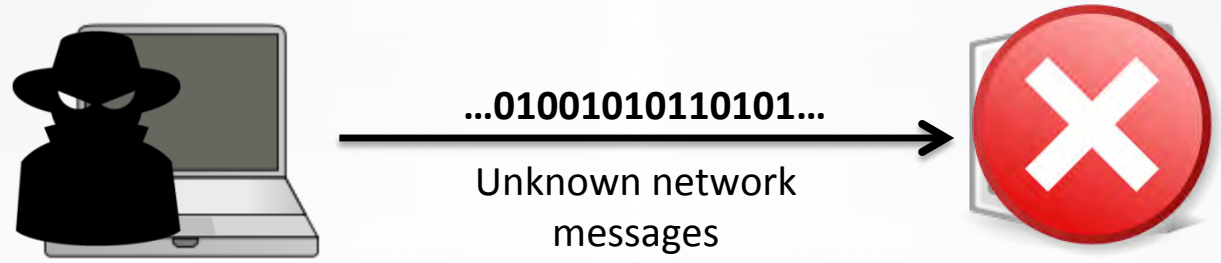
Motivation

- ***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



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

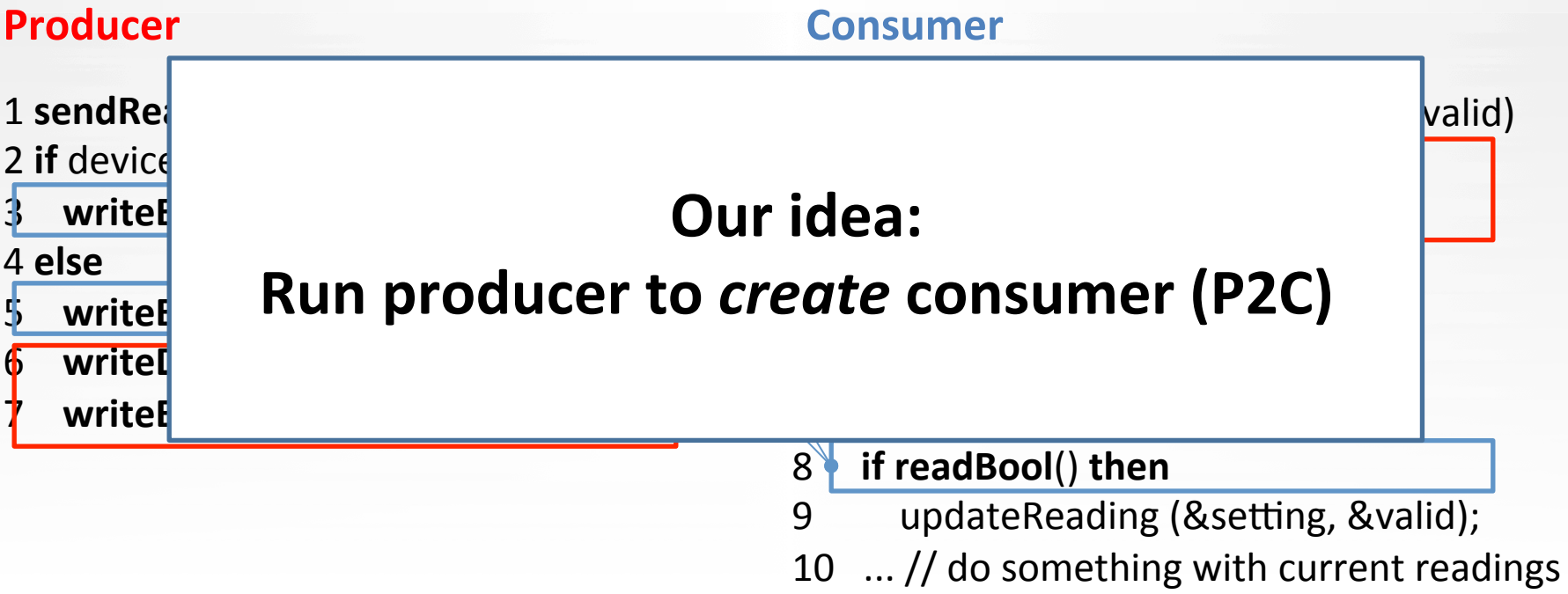
```
1 sendReading(Sensor* device, int prev)
2 if device→setting == prev then
3    writeBool(false);
4 else
5    writeBool(true);
6    writeDouble(device→setting);
7    writeBool(device→valid);
```

Consumer

```
1 updateReading(int* setting, bool* valid)
2    *setting = readDouble();
3    *valid = readBool();
4 main()
5    int setting;
6    bool valid;
7    while ... do
8    if readBool() then
9       updateReading (&setting, &valid);
10   ... // do something with current readings
```

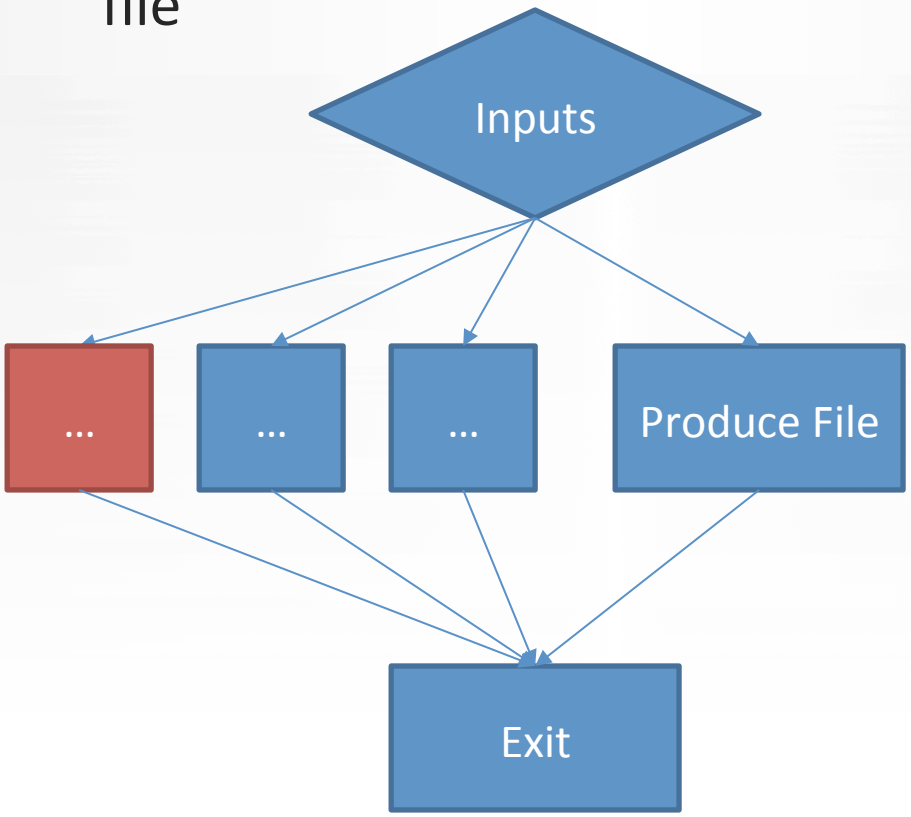
Observation

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



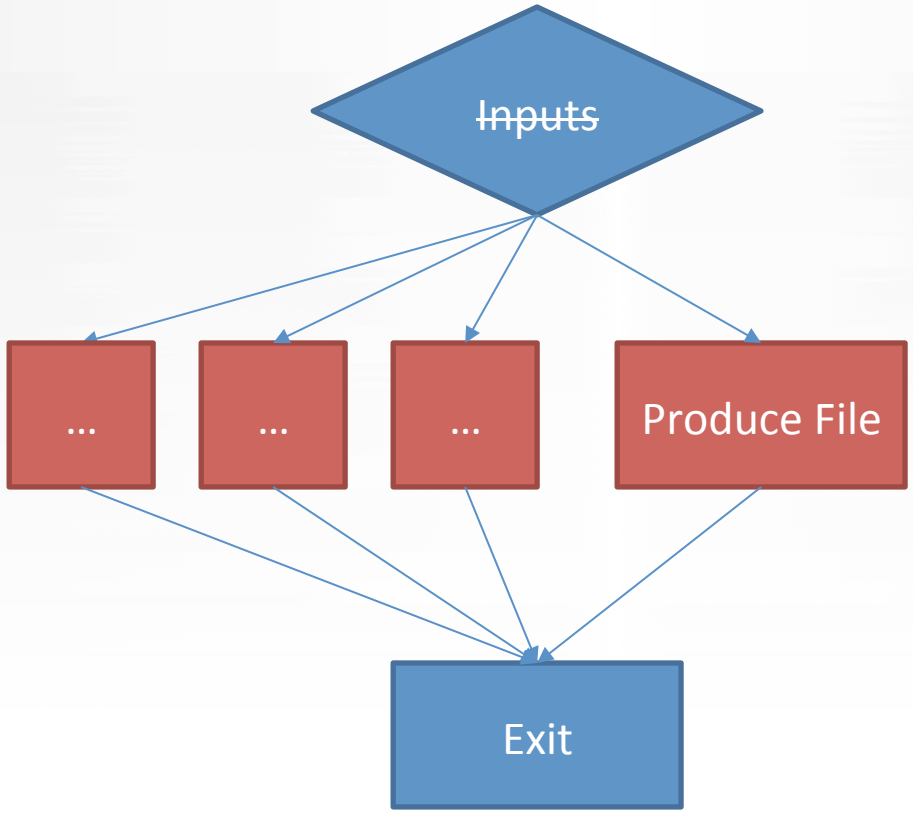
Forced execution

- 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

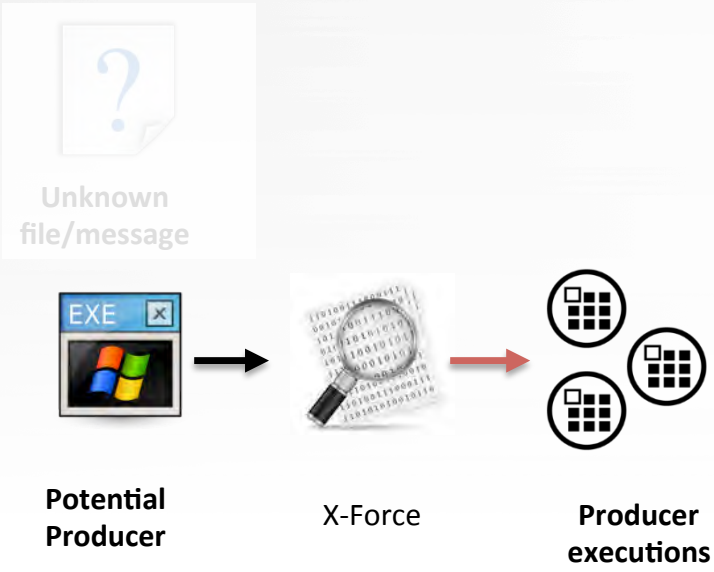
- Basic idea
  - Given an **unknown file/message** and a **potential producer**

Unknown file/message

Potential Producer

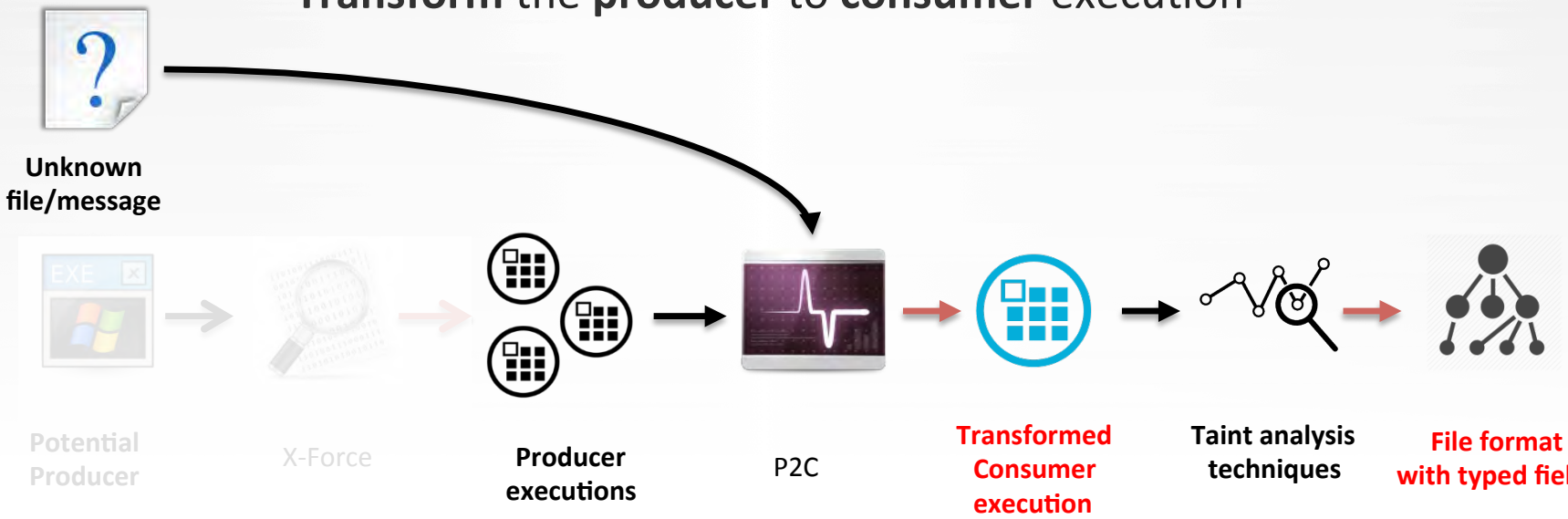
Overview

- 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

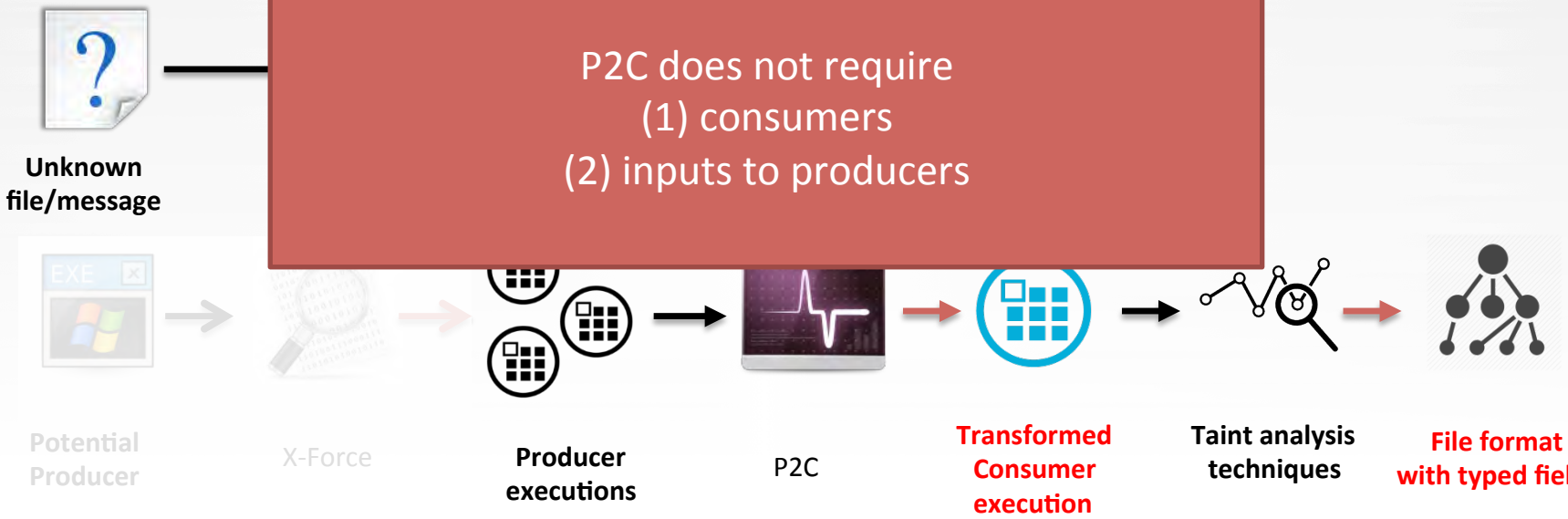
- Basic idea
  - Given an **unknown file/message** and a **potential producer**
  - Explore execution paths that contain "file open-for-write" operations (Producer executions)
    - Leveraging a binary forced execution technique (X-force)
  - **Transform** the **producer** to **consumer** execution



Overview

- Basic idea
  - Given an **unknown file/message** and a **potential producer**
  - Explore execution paths that contain "file open-for-write" operations (Producer executions)
  - -force)
  - P2C directly works on potential producer programs

    P2C does not require
    (1) consumers
    (2) inputs to producers

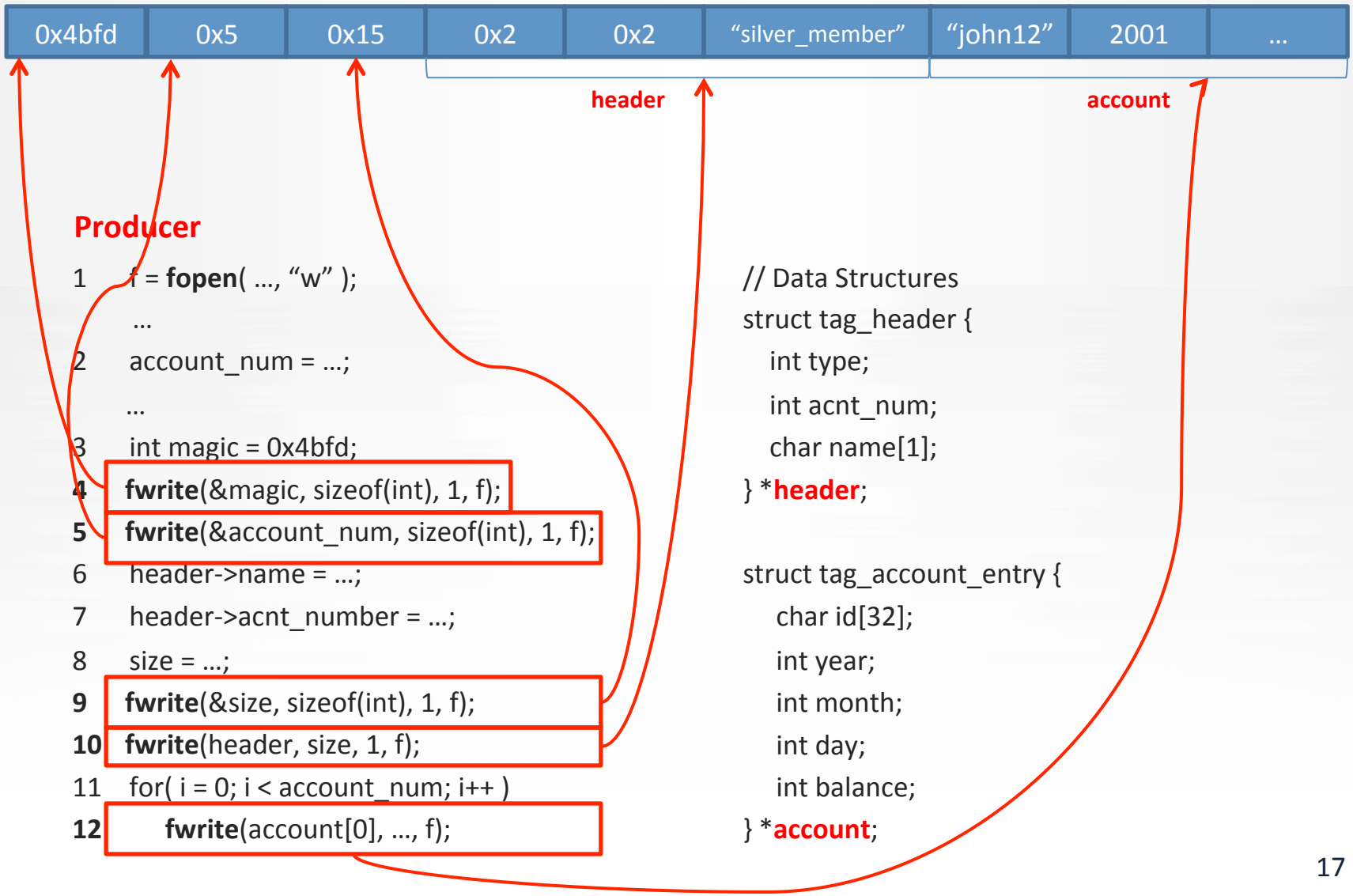


An unknown binary file

0x4bfd	0x5	0x15	0x2	0x2	“silver_member”	“john12”	2001	…

How it is generated?

- **Unknown file**



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, siz
5    fwrite(&account_n
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);
```

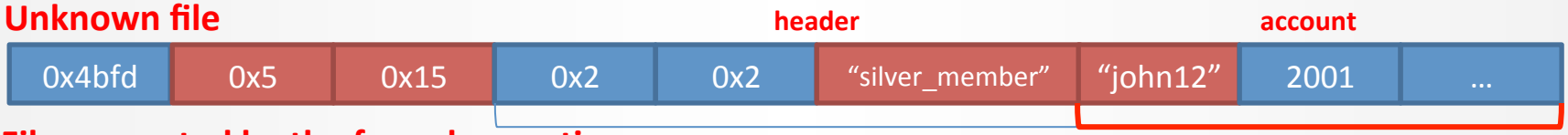
Randomly generated

Randomly generated

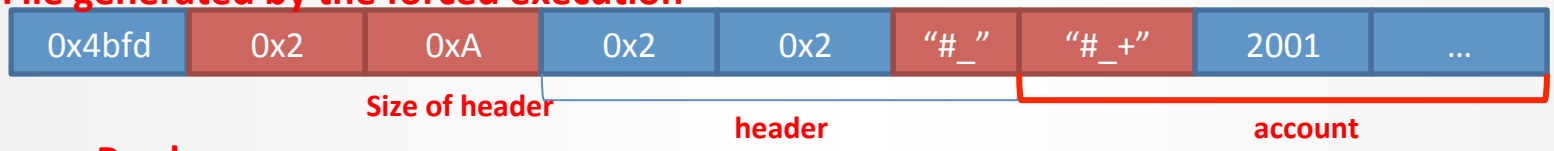
```
// Data Structures
struct tag_header {
   int type;
   int acnt_num;
   char name[1];
} *header;

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**



- **File generated by the forced execution**



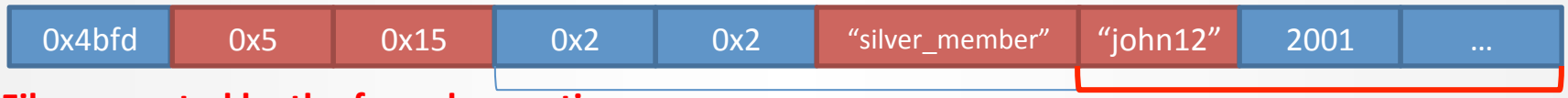
Producer

```
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);
```

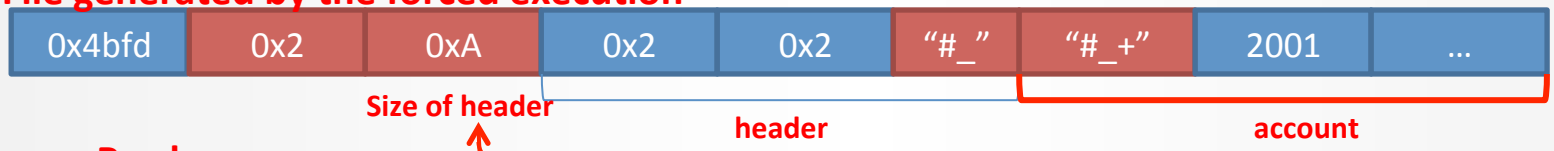
```
// Data Structures
struct tag_header {
   int type;
   int acnt_num;
   char name[1];
} *header;

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

Producer w/o proper inputs (X-force)

- **Unknown file**



- **File generated by the forced execution**



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;
```

Producer w/o proper inputs (X-force)

- **Unknown file**

- **File generated by the forced execution**

Size of header

header

account

Producer

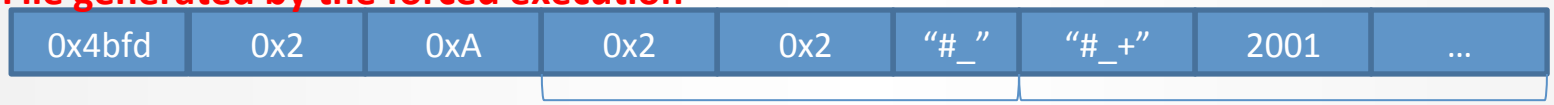
```
1    f = fopen( …, “w” );
     …
2    account_num = …; // 2
     …
3    int magic = 0x4bfd;
4    fwrite(&magic, sizeof(int), 1, f);
5    fwrite(&account_num, sizeof(int), 1, f);
6    header->name = …; // “#_”
7    header->acnt_number = …;  // 2
8    size = …;
9    fwrite(&size, sizeof(int), 1, f);
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;
```

Transforming a producer to a consumer

- **Unknown file**

- **File generated by the forced execution**



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);
```

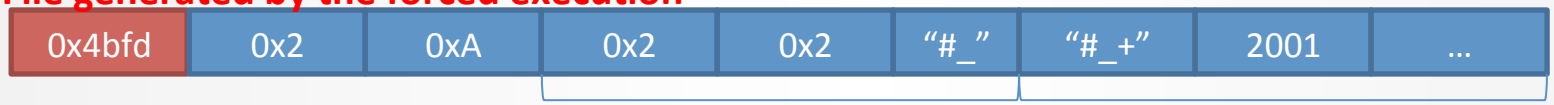
Consumer

```
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**

- **File generated by the forced execution**



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);
```

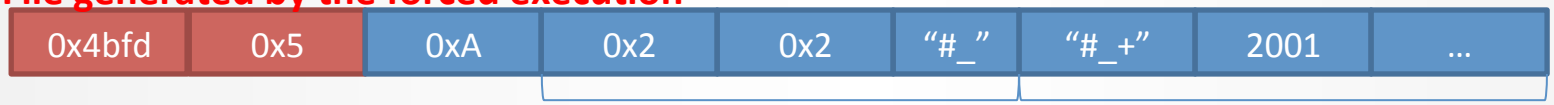
Consumer

```
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**

- **File generated by the forced execution**



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);
```

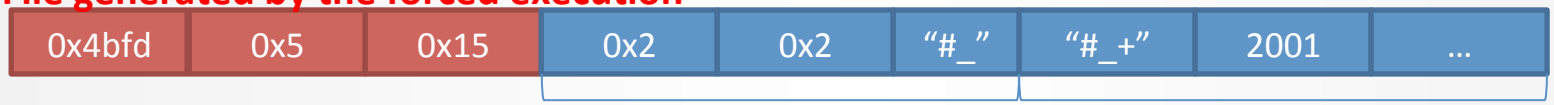
Consumer

```
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**

- **File generated by the forced execution**



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**

- **File generated by the forced execution**

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**

- **File generated by the forced execution**

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);
```

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***

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

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***

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

Evaluation

- ***# of typed fields / total # of fields***

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

Evaluation

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

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

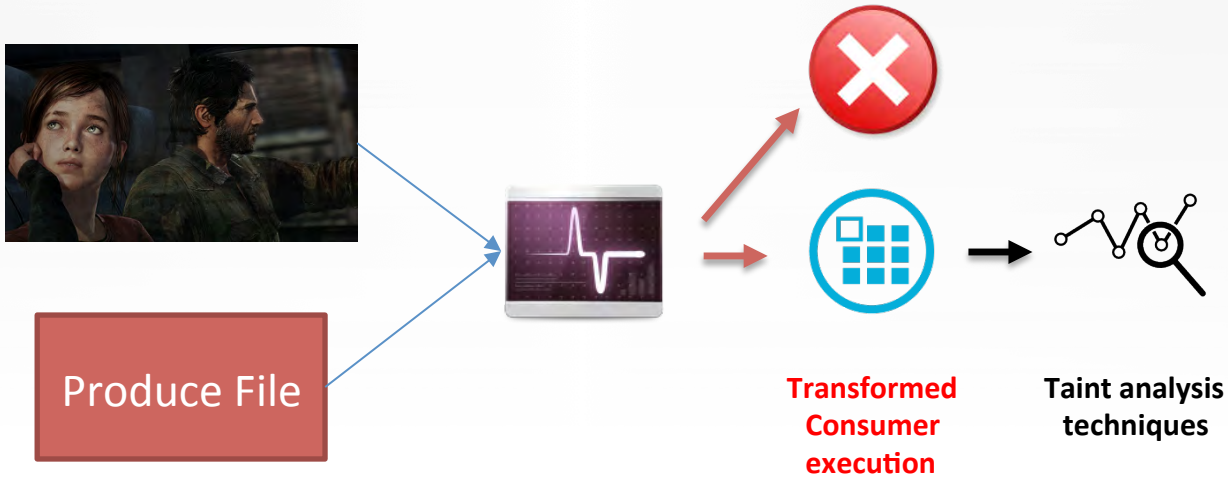
Case study

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

Is there a hidden message or not?

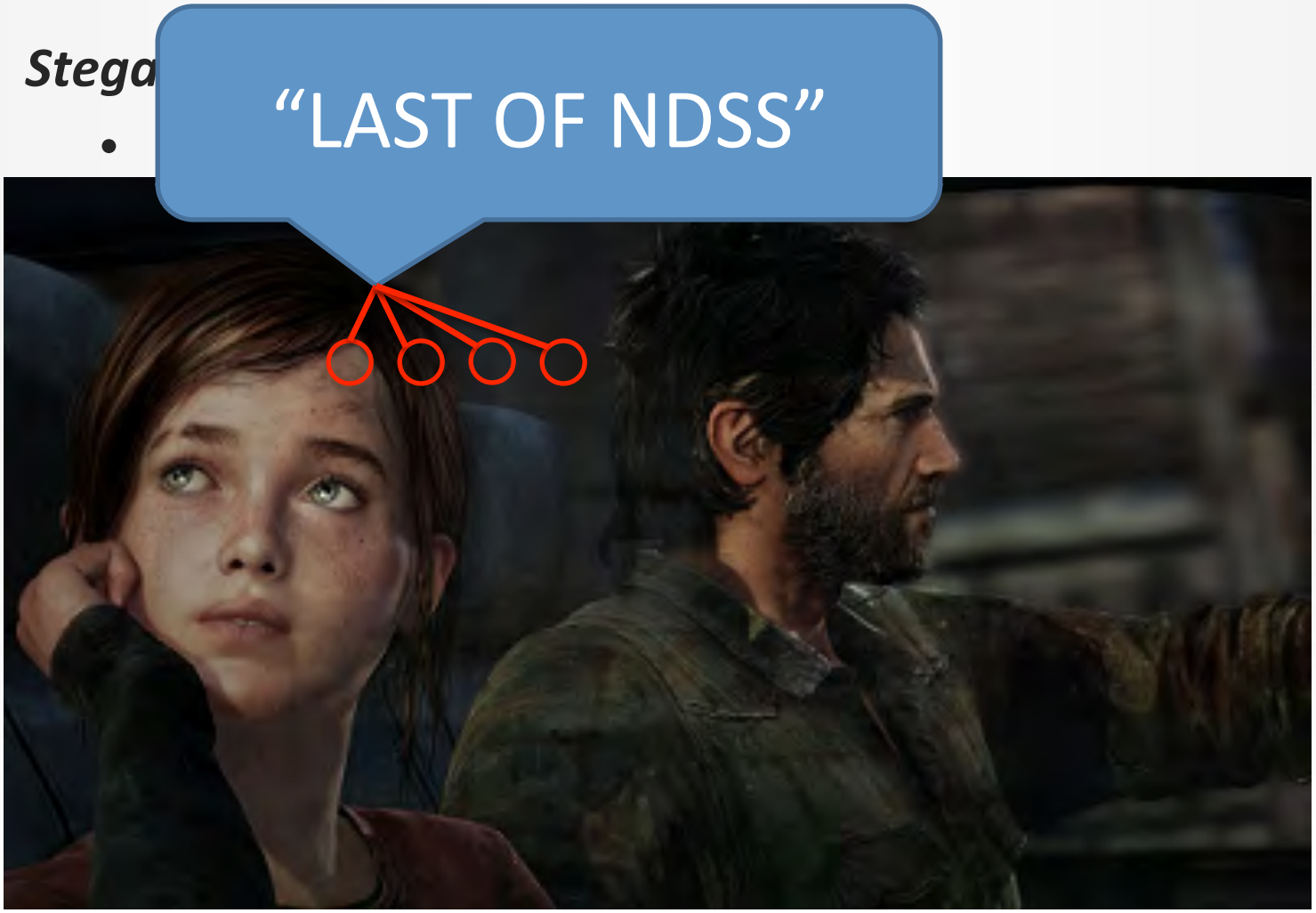
Evaluation

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



Case study

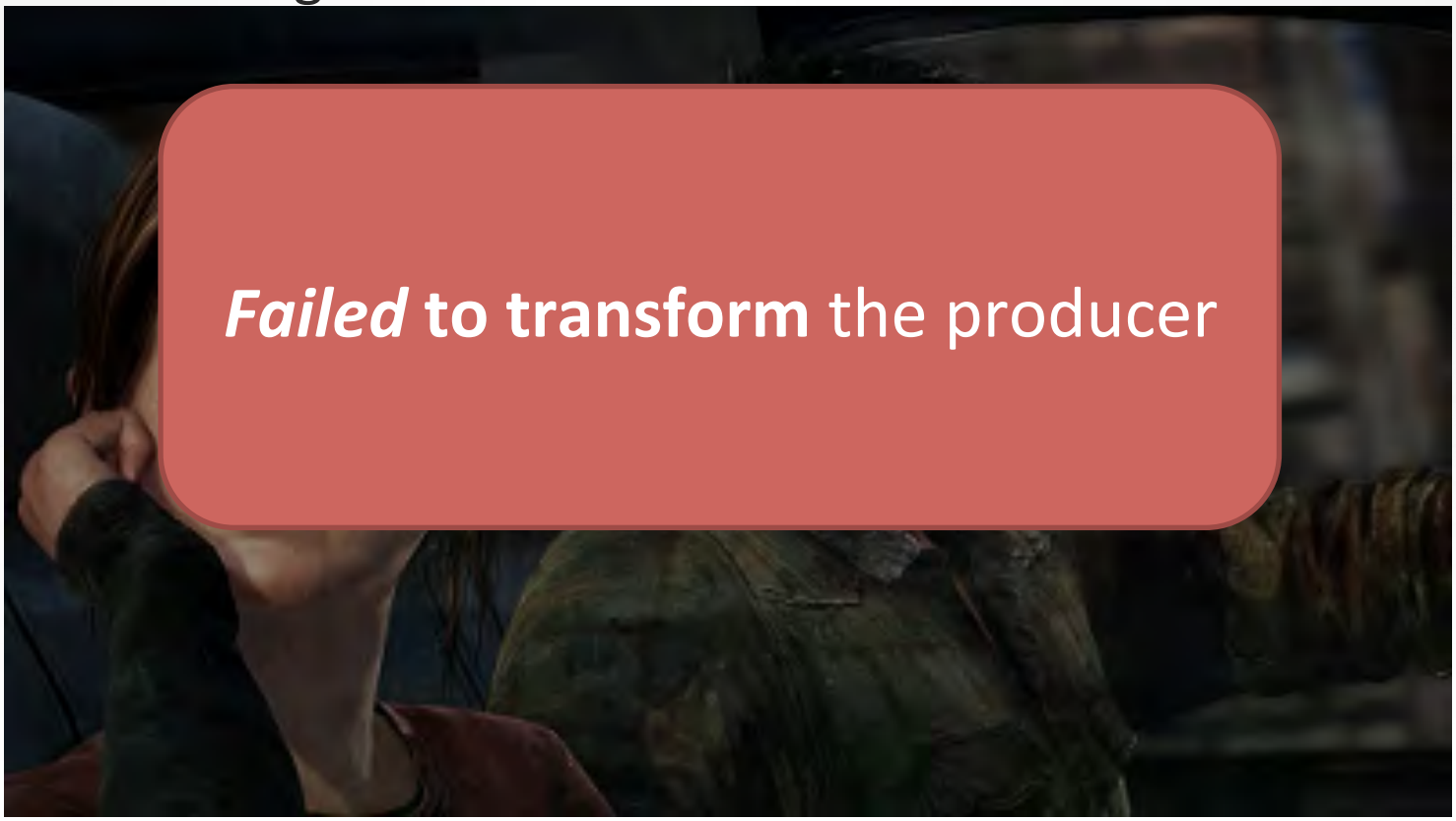
- ***Stega***
  -



Is there a hidden message or not?

Case study

- ***Steganography***
  - Benign



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