Cracking the Odd Case of Randomness in Java

A technique for exploiting insecure randomness in Java with practical applications

elttam

who

Joseph Surin

josep68_ jsur.in

Security Consultant at elttam

- Organises CTFs (with DownUnderCTF)
- Plays CTFs (with skateboarding dog)
- Loves crypto(graphy)

Randomness is everywhere

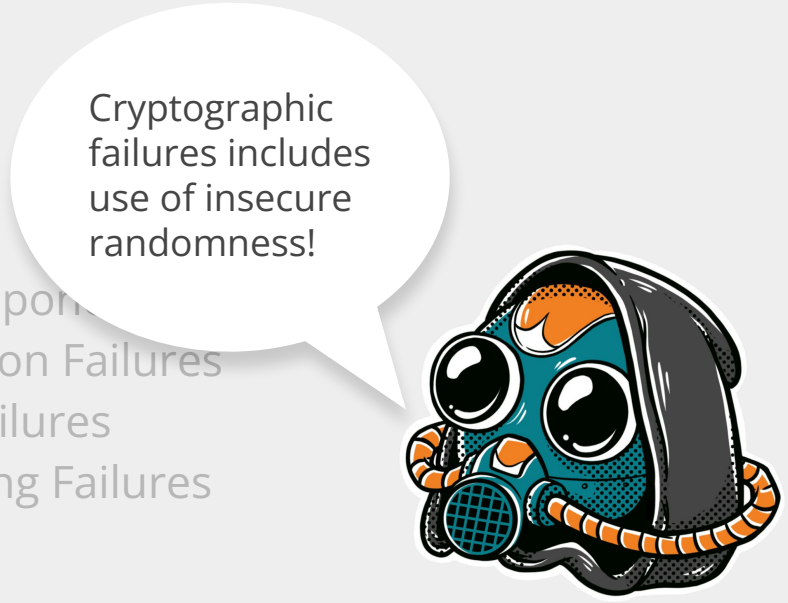
Almost any web application uses randomness for something

- Object IDs
- MFA backup codes
- Session tokens
- Access tokens
- Cryptographic keys/nonces
- Email verification links
- Password reset links
- and so on...

Randomness is not done right everywhere

According to [OWASP Top 10 WebApp Security Risks]

1. A01:2021-Broken Access Control
2. **A02:2021-Cryptographic Failures**
3. A03:2021-Injection
4. A04:2021-Insecure Design
5. A05:2021-Security Misconfiguration
6. A06:2021-Vulnerable and Outdated Compon
7. A07:2021-Identification and Authentication Failures
8. A08:2021-Software and Data Integrity Failures
9. A09:2021-Security Logging and Monitoring Failures
10. A10:2021-Server-Side Request Forgery



Randomness is not done right everywhere

```
String password = RandomStringUtils.random(10, true, true);
```

```
private String verificationCode = RandomStringUtils.randomAlphanumeric(16);
```

```
private String accessToken = RandomStringUtils.randomAlphanumeric(ACCESS_TOKEN_LEN);
```

```
const randomPassword = () => {
  function getRandomSpecialChar() {
    const code = Math.round(Math.random() * (38 - 37) + 37);
    return String.fromCharCode(code);
  }
  function getRandomDigit() {
    const code = Math.round(Math.random() * (57 - 48) + 48);
    return String.fromCharCode(code);
  }
  function getRandomLetter() {
    const code = Math.round(Math.random() * (90 - 65) + 65);
    return String.fromCharCode(code);
  }
  let password = '';
  for (let i = 0; i < 6; i += 1) {
    password += getRandomLetter() + getRandomDigit() + getRandomSpecialChar();
  }
  return password;
};
```

```
def genToken(length=64):
    """
    Use this utility function to generate a random string of
    a desired length.
    """
    return ''.join(random.choice(string.letters + string.digits)
                   for x in range(length))
```

Is it actually exploitable?

Our Target: RandomStringUtils

- Part of Apache Commons Lang 3, a popular Java library providing utility classes
- Very easy to use:

```
String password = RandomStringUtils.randomAlphanumeric(32);
```

Insecure by default!

- Java's default (weak) RNG (`java.util.Random`)


```
36  public class RandomStringUtils {
37
38      /**
39       * <p>Random object used by random method. This has to be not local
40       * to the random method so as to not return the same value in the
41       * same millisecond.</p>
42       */
43      private static final Random RANDOM = new Random();
44
```

org.apache.commons.lang3

Class RandomStringUtils

java.lang.Object
    org.apache.commons.lang3.RandomStringUtils

```
public class RandomStringUtils
extends Object
```

Generates random `Strings`.

Caveat: Instances of `Random`, upon which the implementation of this class relies, are not cryptographically secure.

elttam

RandomStringUtils under the hood

```
140     public static String randomAlphanumeric(final int count) {
141         return random(count, true, true);
142     }
```

```
263     public static String random(final int count, final boolean letters, final
        boolean numbers) {
264         return random(count, 0, 0, letters, numbers);
265     }
```

```
283     public static String random(final int count, final int start, final int end,
        final boolean letters, final boolean numbers) {
284         return random(count, start, end, letters, numbers, null, RANDOM);
285     }
```

elttam

RandomStringUtils under the hood

```
348     public static String random(int count, int start, int end, final boolean letters, final boolean numbers,
349                                 final char[] chars, final Random random) {
350         //[...]
351
352         if (start == 0 && end == 0) {
353             if (chars != null) {
354                 end = chars.length;
355             } else if (!letters && !numbers) {
356                 end = Character.MAX_CODE_POINT;
357             } else { // chars is null, but letters and numbers are both true, so this branch is taken
358                 end = 'z' + 1;
359                 start = ' ';
360             }
361         } else if (end <= start) {
362             throw new IllegalArgumentException("Parameter end (" + end + ") must be greater than start (" + start + ")");
363         }
364
365         //[...]
366
```

elttam

RandomStringUtils under the hood

```
367         final StringBuilder builder = new StringBuilder(count);
368         final int gap = end - start; // gap equals 91
369
370         while (count-- != 0) {
371             final int codePoint;
372             if (chars == null) { // chars is null, so this branch is taken
373                 // since start is 32 and gap is 91, this set codePoint to a random number between 32 and 122 (inclusive)
374                 codePoint = random.nextInt(gap) + start;
375
376                 //[...]
377
378             } else {
379                 //[...]
380             }
381
382             //[...]
383
384             // codepoints in the range between 32 and 122 contains some non-alphanumeric characters, so this branch is not always taken
385             if (letters && Character.isLetter(codePoint)
386                     || numbers && Character.isDigit(codePoint)
387                     || !letters && !numbers) {
388                 builder.appendCodePoint(codePoint);
389
390                 if (numberOfChars == 2) {
391                     count--;
392                 }
393
394             } else {
395                 count++;
396             }
397         }
398         return builder.toString();
```

So we just need to crack `random.nextInt(91)`...

elttam

how can java random nextInt be cracked if the bound is odd?

It is not possible to crack the randomness of `java.util.Random.nextInt()` if the bound is odd, as the algorithm used to generate the random numbers is considered secure. However, if the bound is even, there could be a potential bias in the generated random numbers, which could be exploited to crack the randomness. Therefore, it is recommended to use an odd bound or use a different random number generator such as `java.security.SecureRandom` for more secure random number generation.

elttam

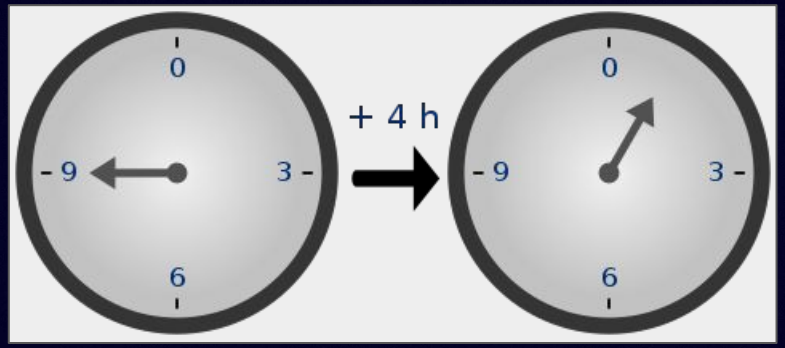
Modular Arithmetic 101

$$a = b \mod M$$

“a is the remainder after dividing b by M”

$$3 = 11 \mod 8$$

$$3 = 11 + 8k, \quad k = -1$$



elttam

java.util.Random 101

- 48-bit Linear Congruential Generator (LCG)
- x_0 is the seed (initial state)
- $x_{i+1} = (Ax_i + C) \mod M$
- `nextInt(bound)` returns an integer less than `bound`
- $y = (x_i \gg 17) \mod B$

```java
protected int next(int bits) {
    AtomicLong seed = this.seed;

    long oldseed;
    long nextseed;
    do {
        oldseed = seed.get();
        nextseed = (oldseed * multiplier + addend) & mask;
    } while(!seed.compareAndSet(oldseed, nextseed));

    return (int)(nextseed >>> 48 - bits);
}
```

```java
public int nextInt(int bound) {
    if (bound <= 0) {
        throw new IllegalArgumentException("bound must be positive");
    } else {
        int r = this.next(31);
        int m = bound - 1;
        if ((bound & m) == 0) {
            r = (int)((long)bound * (long)r >> 31);
        } else {
            for(int u = r; u - (r = u % bound) + m < 0; ) {
                System.out.printf("skipped u = %d\n", u);
                u = this.next(31);
            }
        }

        return r;
    }
}
```

Cracking RandomStringUtils - Setup

- Suppose we call `RandomStringUtils.randomAlphanumeric(10)` and get the string `"Q6p0ifNqAk"`
- Take the ASCII values of each character and subtract 32 to get the outputs of `random.nextInt(91)`
- Call these $y_1, y_2, \ldots, y_{10}$. In this case:

$$y_1 = 49$$

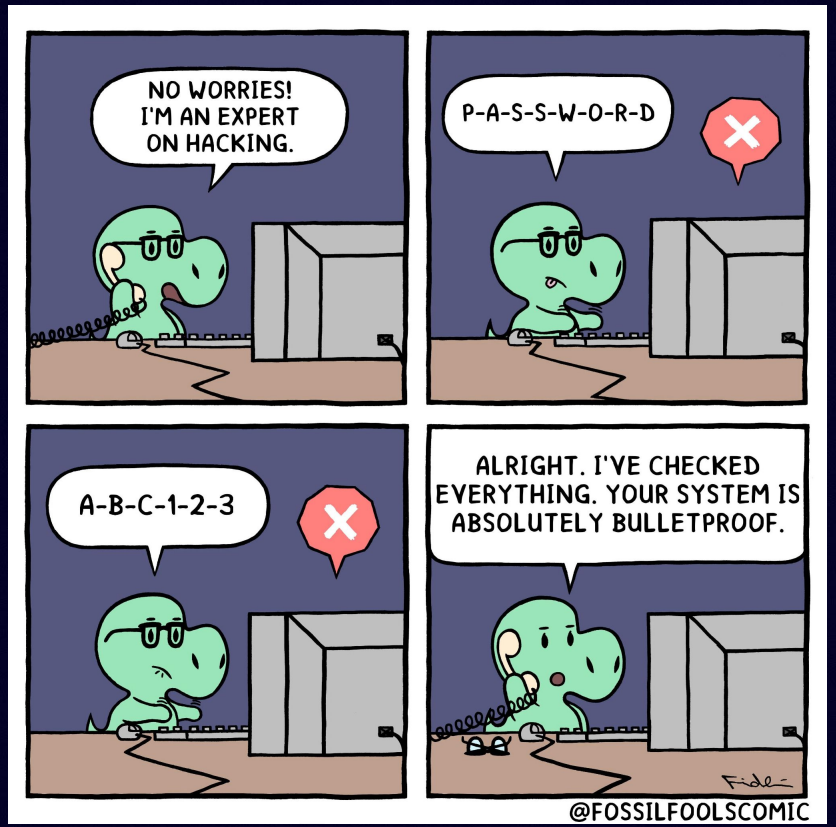
$$y_2 = 22$$

$$y_3 = 80$$

$$\vdots$$

Cracking RandomStringUtils - Idea #0

- When in doubt, use bruteforce
- Try all 2^{48} possible `Random` state values, check which produces a string that agrees with the output
- This will take a while...



Time to stare at equations

elttam

Cracking RandomStringUtils - Idea #1

$$y_1 = (x_1 \gg 17) \mod 91$$

$$\implies y_1 = (x_1 \gg 17) + 91k_1$$

$$\implies y_1 - 91k_1 = (x_1 \gg 17)$$

$$|k_1| < 2^{31}/91$$

Cracking RandomStringUtils - Idea #1

$$y_1 = (x_1 \gg 17) \mod 91$$

$$\implies y_1 = (x_1 \gg 17) + 91k_1$$

$$\implies y_1 - 91k_1 = (x_1 \gg 17)$$

$$|k_1| < 2^{31}/91$$

Cracking RandomStringUtils - Idea #1

$$
\begin{aligned}
& y_1 = (x_1 \gg 17) \mod 91 \\
\implies\ & y_1 = (x_1 \gg 17) + 91k_1 \\
\implies\ & y_1 - 91k_1 = (x_1 \gg 17)
\end{aligned}
$$

$$|k_1| < 2^{31}/91$$

Cracking RandomStringUtils - Idea #1

$$
\begin{aligned}
& y_1 = (x_1 \gg 17) \mod 91 \\
\implies \quad & y_1 = (x_1 \gg 17) + 91k_1 \\
\implies \quad & y_1 - 91k_1 = (x_1 \gg 17) \\
& |k_1| < 2^{31}/91
\end{aligned}
$$

Cracking RandomStringUtils - Idea #1

$$y_1 - 91k_1 = (x_1 \gg 17)$$

Enumerate the $2^{31}/91$ possible values of k_1 to get candidates for $(x_1 \gg 17)$

Cracking RandomStringUtils - Idea #1

$$y_1 - 91k_1 = (x_1 \gg 17)$$

Enumerate the $2^{31}/91$ possible values of k_1 to get candidates for $(x_1 \gg 17)$

Enumerate the 2^{17} possible values of $x_1 \mod 2^{17}$ to get candidates for x_1

Cracking RandomStringUtils - Idea #1

$$y_1 - 91k_1 = (x_1 \gg 17)$$

Enumerate the $2^{31}/91$ possible values of k_1 to get candidates for $(x_1 \gg 17)$

Enumerate the 2^{17} possible values of $x_1 \mod 2^{17}$ to get candidates for x_1

Complexity: $\mathbf{2^{41.5}}$

This is the idea used in [alex91ar/randomstringutils]

Cracking RandomStringUtils - Idea #2

- Idea #0 uses none of the outputs (to reduce the state search space)
- Idea #1 uses one output
- Idea #2 ...

elttam

Time to stare at equations (part 2)

elttam

Cracking RandomStringUtils - Idea #2

$$y_2 = (x_2 \gg 17) \mod 91$$

$$\implies y_2 = (((Ax_1 + C) \mod M) \gg 17) \mod 91$$

$$\implies y_2 = (((A(x_{1,U_{31}} + x_{1,L_{17}}) + C) \mod M) \gg 17) \mod 91$$

$$x_1 = \boxed{x_{1,U_{31}} \mid x_{1,L_{17}}}$$

Cracking RandomStringUtils - Idea #2

$$y_2 = (x_2 \gg 17) \mod 91$$

$$\implies y_2 = (((Ax_1 + C) \mod M) \gg 17) \mod 91$$

$$\implies y_2 = (((A(x_{1,U_{31}} + x_{1,L_{17}}) + C) \mod M) \gg 17) \mod 91$$

$$x_1 = \boxed{x_{1,U_{31}} \mid x_{1,L_{17}}}$$

Cracking RandomStringUtils - Idea #2

$$y_2 = (x_2 \gg 17) \mod 91$$

$$\implies y_2 = (((Ax_1 + C) \mod M) \gg 17) \mod 91$$

$$\implies y_2 = (((A(x_{1,U_{31}} + x_{1,L_{17}}) + C) \mod M) \gg 17) \mod 91$$

$$x_1 = \boxed{x_{1,U_{31}} \mid x_{1,L_{17}}}$$

Cracking RandomStringUtils - Idea #2

$$y_2 = (((Ax_{1,U_{31}} + Ax_{1,L_{17}} + C) \mod M) \gg 17) \mod 91$$

$$\implies y_2 = ((((Ax_{1,U_{31}} + C) \mod M) + (Ax_{1,L_{17}} \mod M)) \mod M) \gg 17) \mod 91$$

$$\implies y_2 = ((((Ax_{1,U_{31}} + C) \mod M) + (Ax_{1,L_{17}} \mod M) - b_2 M) \gg 17) \mod 91$$

$$b_2 \in \{0, 1\}$$

Cracking RandomStringUtils - Idea #2

$$y_2 = (((Ax_{1,U_{31}} + Ax_{1,L_{17}} + C) \mod M) \gg 17) \mod 91$$

$$\implies y_2 = (((((Ax_{1,U_{31}} + C) \mod M) + (Ax_{1,L_{17}} \mod M)) \mod M) \gg 17) \mod 91$$

$$\implies y_2 = ((((Ax_{1,U_{31}} + C) \mod M) + (Ax_{1,L_{17}} \mod M) - b_2 M) \gg 17) \mod 91$$

$$b_2 \in \{0, 1\}$$

Cracking RandomStringUtils - Idea #2

$$y_2 = (((Ax_{1,U_{31}} + Ax_{1,L_{17}} + C) \mod M) \gg 17) \mod 91$$

$$\implies y_2 = (((((Ax_{1,U_{31}} + C) \mod M) + (Ax_{1,L_{17}} \mod M)) \mod M) \gg 17) \mod 91$$

$$\implies y_2 = ((((Ax_{1,U_{31}} + C) \mod M) + (Ax_{1,L_{17}} \mod M) - b_2 M) \gg 17) \mod 91$$

$$b_2 \in \{0, 1\}$$

Cracking RandomStringUtils - Idea #2

$$y_2 = ((((Ax_{1,U_{31}} + C) \mod M) + (Ax_{1,L_{17}} \mod M) - b_2 M) \gg 17) \mod 91$$

$$\Longrightarrow y_2 = ((((Ax_{1,U_{31}} + C) \mod M) \gg 17) + ((Ax_{1,L_{17}} \mod M) \gg 17) - b_2 2^{31} + c_2) \mod 91$$

$$c_2 \in \{0, 1\}$$

Cracking RandomStringUtils - Idea #2

$$y_2 = ((((Ax_{1,U_{31}} + C) \mod M) + (Ax_{1,L_{17}} \mod M) - b_2 M) \gg 17) \mod 91$$

$$\implies y_2 = ((((Ax_{1,U_{31}} + C) \mod M) \gg 17) + ((Ax_{1,L_{17}} \mod M) \gg 17) - b_2 2^{31} + c_2) \mod 91$$

$$c_2 \in \{0, 1\}$$

Cracking RandomStringUtils - Idea #2

$$y_2 = ((((Ax_{1,U_{31}} + C) \mod M) \gg 17) + ((Ax_{1,L_{17}} \mod M) \gg 17) - b_2 2^{31} + c_2) \mod 91$$

$$\implies (y_2 - (((Ax_{1,U_{31}} + C) \mod M) \gg 17)) \mod 91 = (((Ax_{1,L_{17}} \mod M) \gg 17) - b_2 2^{31} + c_2) \mod 91$$

From idea #1, we have $2^{31}/91$ candidates for $x_{1,U_{31}}$ ...

Cracking RandomStringUtils - Idea #2

$$y_2 = ((((Ax_{1,U_{31}} + C) \mod M) \gg 17) + ((Ax_{1,L_{17}} \mod M) \gg 17) - b_2 2^{31} + c_2) \mod 91$$

$$\implies (y_2 - (((Ax_{1,U_{31}} + C) \mod M) \gg 17)) \mod 91 = (((Ax_{1,L_{17}} \mod M) \gg 17) - b_2 2^{31} + c_2) \mod 91$$

From idea #1, we have $2^{31}/91$ candidates for $x_{1,U_{31}}$ ...

Cracking RandomStringUtils - Idea #2

$$y_2 = ((((Ax_{1,U_{31}} + C) \mod M) \gg 17) + ((Ax_{1,L_{17}} \mod M) \gg 17) - b_2 2^{31} + c_2) \mod 91$$

$$\implies (y_2 - (((Ax_{1,U_{31}} + C) \mod M) \gg 17)) \mod 91 = (((Ax_{1,L_{17}} \mod M) \gg 17) - b_2 2^{31} + c_2) \mod 91$$

From idea #1, we have $2^{31}/91$ candidates for $x_{1,U_{31}}$ ...

Cracking RandomStringUtils - Idea #2

Compute a lookup table for left-hand side values

$$(y_2 - (((Ax_{1,U_{31}} + C) \mod M) \gg 17)) \mod 91$$

to $x_{1,U_{31}}$ candidates

For each $x_{1,L_{17}}$ candidate, compute the right-hand side value

$$(((Ax_{1,L_{17}} \mod M) \gg 17) - b_2 2^{31} + c_2) \mod 91$$

and meet-in-the-middle!

elttam

Cracking RandomStringUtils - Idea #2

- Approximately $2^{31}/91^2$ candidates for $x_{1,U_{31}}$ for each left-hand side value

Cracking RandomStringUtils - Idea #2

- Approximately $2^{31}/91^2$ candidates for $x_{1,U_{31}}$ for each left-hand side value
- 2^{19} right-hand side values to go through

Cracking RandomStringUtils - Idea #2

- Approximately $2^{31}/91^2$ candidates for $x_{1,U_{31}}$ for each left-hand side value
- 2^{19} right-hand side values to go through
- Complexity: **2^{37}**

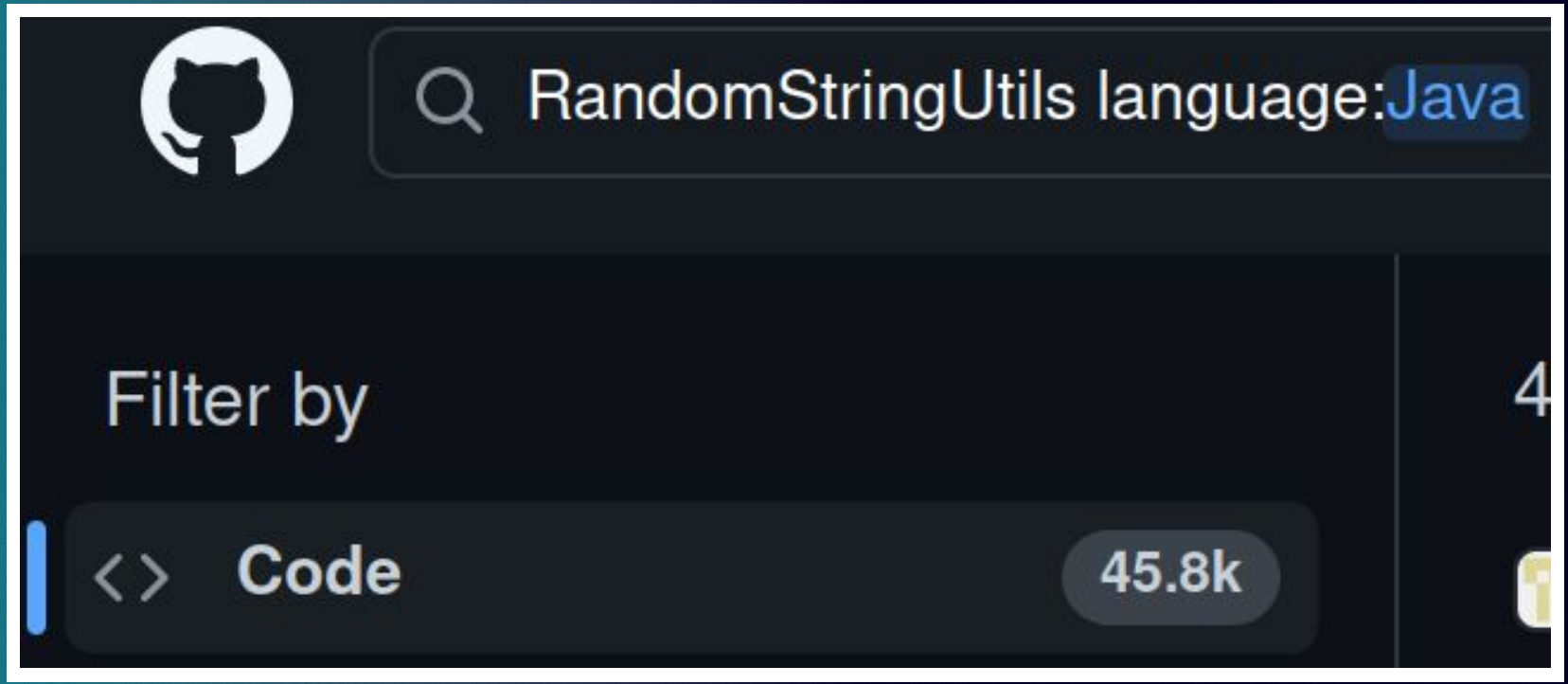
Cracking RandomStringUtils - Idea #2

- Approximately $2^{31}/91^2$ candidates for $x_{1,U_{31}}$ for each left-hand side value
- 2^{19} right-hand side values to go through
- Complexity: **2^{37}**
- Idea can be generalised, using another output gives a time complexity improvement of 4.5 bits

Cracking RandomStringUtils - Idea #2

- Approximately $2^{31}/91^2$ candidates for $x_{1,U_{31}}$ for each left-hand side value
- 2^{19} right-hand side values to go through
- Complexity: **2^{37}**
- Idea can be generalised, using another output gives a time complexity improvement of 4.5 bits
- Full details: https://elttam.com/blog/cracking-randomness-in-java/

That’s cool but who cares?



elttam

```
mbok.extern.slf4j.Slf4j;
g.apache.commons.lang3.RandomStringUtils;
g.springframework.boot.context.event.ApplicationReadyEvent;

// generate password
password = RandomStringUtils.randomAlphanumeric(16);
log.info("=== Generated random password: {} for super administrator: {} ===",
```

```
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.StringUtils;

if (StringUtils.isBlank(cookieSecret)) {
    cookieSecret = RandomStringUtils.randomAlphanumeric(
}
```

```
org.kurento.client.GenericMediaElement;

String rtspConnectionId = kMed
        + RandomStringUtils.randomAlphanume
        + (uri.getPort() != -1 ? (":" + uri.getPort()) : "") + uri.getPath();
```

```
566
567
    if(listOfDisabledFeatures.contains("learningDashboard") == false) {
        learningDashboardAccessToken = RandomStringUtils.randomAlphanumeric
    }
```

```
import org.apache.commons.lang.RandomStringUtils;

    private static String _getPassword ( String key, int length ) {
        return RandomStringUtils.random( length, key );
    }
```

```
41
42    private static final String KEY_STORE_PASS = RandomStringUtils.randomAlphanumeric(8);
43    private static final String KEY_STORE_FILE_PATH = "clientkeystore.jks";
```

ona › app/models/Email.java

```
port controllers.routes;
port org.apache.commons.lang3.RandomStringUtils;
port play.data.validation.Constraints;

public void sendValidationEmail() {
    this.token = RandomStringUtils.randomNumeric(50);
    this.confirmUrl = Url.create(routes.UserApp.confirmEmail(this.id, this.token).url());
```

t-Media-Server › src/main/java/io/antmedia/filter/TokenGenerator.java

```
mport org.apache.commons.lang3.RandomStringUtils;

if(genetaredToken == null) {
    genetaredToken = RandomStringUtils.random
```

Demo

(CVE-2023-24828)

Found with Emily Trau (emily.id.au)

elttam

Practical Applications

- Developing exploits quickly
- Detecting insecure randomness in black-box settings

Conclusion

- Insecure randomness ⇒ easily exploitable, high risk issues
- Crypto is fun
- Tool available at https://github.com/elttam/rsu-cracker
- Full details + other awesome posts at https://elttam.com/blog

Thank You!

Questions?

elttam