



Aa



# Breaking the NemucodAES Ransomware

The Nemucod ransomware has been around, in various incarnations, for some time. Recently a new variant started spreading via email claiming to be from UPS. This new version changed how files are encrypted, clearly in an attempt to fix its prior issue of being able to decrypt files without paying the ransom, and as this is a new version, no decryptor was available<sup>1</sup>. My friends at [Savage Security](#) contacted me to help save the data of one of their clients; I immediately began studying the cryptography related portions of the software, while the Savage Security team was busy looking at other portions.

## The Code #

The code that really matters is in a PHP file<sup>2</sup>, named after the Bitcoin address that the victim is to pay the ransom to, and stored under the user's %TEMP% directory. Here's the bit that matters to us:

```
if ($stat_files > 0) {  
    $db = fopen($fn . ".db", "w");  
    foreach ($_SERVER["files"] as $file) {  
        $fp = fopen($file, "r+");  
        if ($fp === false) continue;  
        $trash = "";  
        for ($i = 0; $i < 2048; $i++) $trash.= chr(mt_rand(0, 255));  
        $key = "";  
        for ($i = 0; $i < 128; $i++) $key.= chr(mt_rand(0, 255));  
        $aes = new Crypt_AES(CRYPT_AES_MODE_ECB);  
        $aes->setKeyLength(128);  
        $aes->setKey($key);
```

```

$b = fread($fp, 2048);
fseek($fp, 0);
fwrite($fp, substr($trash, 0, strlen($b)));
fclose($fp);
$b = $aes->encrypt($b);
$rsa = new Crypt_RSA();
$rsa->loadKey($keypub);
$key = $rsa->encrypt($key);
fputs($db, $file . " " . base64_encode($key) . " " . base64_enc
");

```




There are some important things that we see immediately:

- They generate a unique encryption key for each file.
- They are using AES-128 in **ECB** mode.
- They are using RSA to encrypt the AES-128 key and store it in a `.db` file (also named after the Bitcoin address).
- They encrypt the first 2,048 bytes of the file, and then replace it with random data.
- The `.db` file contains the path, encrypted AES-128 key, and the encrypted data removed from the file.

## The Critical Mistake(s) #

If you've been to any of my talks on cryptography, you should see an immediate issue with this code. If not, let me point this line out:

```
for ($i = 0;$i < 128;$i++) $key.= chr(mt_rand(0, 255));
```

This line creates a 128 **byte** key to be used to encry  Add to Firefox - it's free

developers don't know bits from bytes), using PHP's `mt_rand` function. This function generates random numbers using the [Mersenne Twister](#) algorithm, which happens to use a small (32-bit) seed – this is where the fun begins.



Because of this small seed, if we can observe the initial output of `mt_rand`, we can brute-force the seed and then predict its future output. Thankfully, the developers of Nemucod made this easy for us. If you recall, the first 2,048 bytes of each file are replaced with random data from `mt_rand`, then the encryption key is generated immediately after. This means that they have given us everything we need.

Using the first few bytes (4 or 5), we can brute-force the seed that `mt_rand` used<sup>3</sup>, and by running `mt_rand` the appropriate number of times, we can create the exact output that the PHP script did when it encrypted the files, revealing the file encryption keys and allowing us to decrypt all of the files.

## Cracking the Seed #

To get the seed, we need to brute-force all  $2^{32}$  possible values, thankfully there's a handy tool to do this – and do it within a matter of seconds. A few years ago the always impressive [Solar Designer](#) released just what [we need](#). This is a simple command-line tool that takes output (in this case the first few bytes of the first file encrypted) and provides the seed that was used.

```
./php_mt_seed 98 98 0 255 251 251 0 255 47 47 0 255 131 131 0 255
Pattern: EXACT-FROM-256 EXACT-FROM-256 EXACT-FROM-256 EXACT-FROM-256
Found 0, trying 1241513984 - 1275068415, speed 39053601 seeds per second
seed = 1241912029
Found 1, trying 4261412864 - 4294967295, speed 4555776 seeds per second
```



Using `php_mt_seed`, it takes only about a minute to test all of the possible seeds, and identify the correct one. Once we have that, decryption is simple, and we have all of the data back without paying a single cent to the extortionists.

## Why Randomness Matters #

When it comes to key generation (and many other aspects of cryptography), the use of a secure random number generator is critical. If you look at the [documentation](#) for `mt_rand`, you'll see this very clear warning:

This function does not generate cryptographically secure values, and should not be used for cryptographic purposes. If you need a cryptographically secure value, consider using `random_int()`, `random_bytes()`, or `openssl_random_pseudo_bytes()` instead.

Had the developers heeded this warning, and used a more appropriate method for generating the file encryption keys, this method would not have worked. Had the developers not been so kind as to provide us with output from `mt_rand` in the files, this would not have worked. It is the developers of Nemucod that made recovering the data trivial, due to the lack of understanding of proper secure techniques<sup>4</sup>. While I don't want to aid ransomware authors, this is a well known aspect of cryptography – if you write crypto code without a full understanding of what you are doing, and what you are using, this is what happens.

---

1. In the hours before this post was published, Err.

decryption tool for those hit by this version of Nemucod. ↩

2. This ransomware targets Windows users, though the core is written in PHP. The downloader, written in JavaScript, downloads the Windows version of PHP (version 5.6.3) and uses that to execute the PHP file.

Yes, this is as crazy as it sounds. ↩

3. If `mt_rand` is not seeded explicitly via `mt_srand`, PHP will select a random seed, and seed it automatically. In this case, the developers did not explicitly select a seed. ↩

4. There may be additional methods here that could be used, but those are beyond the scope of this article. ↩