Decrypting L0rdix RAT's C2 | HP Wolf Security

By Alex Holland

Published: 2019-08-02 · Archived: 2026-04-05 13:37:28 UTC

In my previous blog post on L0rdix RAT, I took a look at its panel and builder components that have been circulating through underground forums recently. As part of that analysis, I identified a key ("3sc3RLrpd17") which was embedded in one of the PHP pages in L0rdix's panel. A SHA-256 hash is calculated of this key, which is used as the AES key to encrypt and decrypt L0rdix's command and control (C2) communications. When a sample is generated using L0rdix's builder, the operator is able to decide this key.

In this post, I examine L0rdix's C2 encryption and decryption functions in more detail and discuss how to automate the task of identifying, decrypting and extracting L0rdix C2 traffic from a PCAP using Python.

L0rdix's configuration structure

L0rdix's configuration contains 10 fields, which are encrypted and sent as URL query strings in a HTTP POST request to the *connect.php* page of the panel. The configuration settings of deployed bots are updated by sending similar POST requests to the bots from the panel.

Query String	Configuration Field
h=	Hardware ID
o=	Operating system
c=	CPU
g=	GPU
w=	Installed antivirus
p=	Privileges of current user
r=	Hash rate
f=	L0rdix profile in use
rm=	RAM
d=	Drives

L0rdix C2 encryption and decryption steps

L0rdix encrypts its C2 communications using the following steps:

1. Encrypts the plaintext using AES in Cipher Block Chaining (CBC) mode with a 256-bit key and 16-byte initialisation vector (IV).
2. Base64 encodes the ciphertext.
3. Replaces plus (+) characters with tildes (~).
4. URL encodes the ciphertext.

```
public static string encrypt(string plainText)
{
    Aes aes = Aes.Create();
    aes.Mode = CipherMode.CBC;
    SHA256 sha = SHA256.Create();
    aes.Key = sha.ComputeHash(Encoding.ASCII.GetBytes(Config.key));
    aes.IV = new byte[16];
    MemoryStream memoryStream = new MemoryStream();
    ICryptoTransform transform = aes.CreateEncryptor();
    CryptoStream cryptoStream = new CryptoStream(memoryStream, transform, CryptoStreamMode.Write);
    byte[] bytes = Encoding.ASCII.GetBytes(plainText);
    cryptoStream.Write(bytes, 0, bytes.Length);
    cryptoStream.FlushFinalBlock();
    byte[] array = memoryStream.ToArray();
    memoryStream.Close();
    cryptoStream.Close();
    return Convert.ToBase64String(array, 0, array.Length).Replace("+", "~");
}
```

Figure 1 – L0rdix RAT's C# encryption function.

For example, the ciphertext of "Windows 7 Enterprise" looks like this:

- *buNpZksa9PSEshjHiM9XNI84ku2X6Zy2Syr7zdzvxMM%3d*

We can decrypt the ciphertext by performing the encryption actions in reverse:

- *buNpZksa9PSEshjHiM9XNI84ku2X6Zy2Syr7zdzvxMM=* (After URL decoding)
- *6ee369664b1af4f484b218c788cf57348f3892ed97e99cb64b2afbcddcefc4c3* (Hex representation after Base64 decoding)
- *Windows 7 Enterprise* (After AES decryption)

Default key or single operator

After analysing more L0rdix samples in the wild, it became clear that many of them use the key found in the leaked panel to encrypt their C2 channels. There are two realistic possibilities for the re-occurrence of this key:

1. "3sc3RLrpd17" is the default key that the RAT panel is supplied with from its author and several buyers have not changed it.
2. The L0rdix samples that use this key are bots controlled by a single operator. Based on the implementation of the encryption and decryption functions, each L0rdix panel operator must use the same key for every bot they control.

There is stronger evidence to indicate that the first possibility is true. For instance, the key is referenced in a coding tutorial on how to encrypt and decrypt data using AES in C# and PHP. L0rdix's server side encryption and decryption functions closely resemble those in the tutorial, sharing the method (AES-CBC 256), IV (16 null

bytes), key (“3sc3RLrpd17”) and programming languages used (C# and PHP). L0rdix’s author may have copied the tutorial’s encryption implementation and decided not to change the key.

```
define(KEY, "3sc3RLrpd17");  <-- Default C2 encryption key

function decrypt($encrypted){
    $method = 'aes-256-cbc';

    $encrypted = str_replace("~", "+", $encrypted);
    $password = substr(hash('sha256',  KEY ,true), 0, 32);

    $iv = chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0)
     . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0);


    $decrypted = openssl_decrypt(base64_decode($encrypted), $method, $password,
    OPENSSL_RAW_DATA, $iv);
    return $decrypted;
}
```

Figure 2 – L0rdix panel’s PHP decryption function, including the suspected default operator key.

```
$plaintext = 'My secret message 1234';
$password = '3sc3RLrpd17';

// CBC has an IV and thus needs randomness every time a message is encrypted
$method = 'aes-256-cbc';

// Must be exact 32 chars (256 bit)
// You must store this secret random key in a safe place of your system.
$key = substr(hash('sha256', $password, true), 0, 32);
echo "Password:" . $password . "\n";

// Most secure key
//$key = openssl_random_pseudo_bytes(openssl_cipher_iv_length($method));

// IV must be exact 16 chars (128 bit)
$iv = chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) . chr(0x0) .
```

Figure 3 – Coding tutorial on AES-256 encryption and decryption in PHP.

Automating L0rdix C2 decryption with decrypt_l0rdix_c2.py

Since L0rdix uses symmetric key encryption it was possible to write a Python script (decrypt_l0rdix_c2.py) to parse a PCAP containing L0rdix C2 traffic and decrypt it. The script first identifies L0rdix traffic based on its expected structure by parsing a PCAP using the Pyshark library. For example, since L0rdix has 10 fields in its configuration, there should be at least 10 query strings in a L0rdix C2 URL. The query strings are identified and decoded into a ciphertext before being run through an AES decryption function using the Pycryptodome library.

The script also extracts screenshots that L0rdix RAT periodically takes of infected systems. The traffic is decrypted using the suspected default operator key (“3sc3RLrpd17”), or a user supplied key using the optional -k argument. You can extract the operator key from a L0rdix bot using static analysis. If a L0rdix operator has not changed the operator key from the default one, any captured C2 traffic between their bots and the admin panel can be decrypted using decrypt_l0rdix_c2.py. The script is available to download from GitHub.

Figures 4 and 5 show the output from running the script against L0rdix C2 traffic:

- *decrypt_l0rdix_c2.py -p l0rdix_c2.pcap*

```
[+] Parsing PCAP...
[+] Searching for L0rdix C2 traffic...
[+] Found references to L0rdix C2 servers (9):

file4.pw
file4.pw
file4.pw
file4.pw
file4.pw
file4.pw
file4.pw
file4.pw
file4.pw

[+] Found L0rdix C2 traffic (90 strings):

34LOZPCw/mPI5HUy4Lsi0Q==
buNpZksa9PSEshjHiM9XNI84ku2X6Zy2Syr7zdzvxMM=
3aTb7srDzYxEWFkQnwnJ7g8/q1lMRTPDwq4BGCcqe4NCOLdrYH3nO3AN1IYu3lUX
XCqvQT7cakcYT45f7GW8w/6BFBjSKY31PWyuNGKwrfQ=
C84qIwUxZ8YuMnqgeC+lVQ==
R3F57LtVGns08zxFybyhTA==
P/qrhb+vh8Et4Myw8AgxpA==
8oP293sYGZNlvxI9VYwnHw==
3B229kg7neneBTT7kKJmCA==
fd+s7G8AWjiF7OXsYIn0Hg==
34LOZPCw/mPI5HUy4Lsi0Q==
buNpZksa9PSEshjHiM9XNI84ku2X6Zy2Syr7zdzvxMM=
3aTb7srDzYxEWFkQnwnJ7g8/q1lMRTPDwq4BGCcqe4NCOLdrYH3nO3AN1IYu3lUX
XCqvQT7cakcYT45f7GW8w/6BFBjSKY31PWyuNGKwrfQ=
```

Figure 4 – Output from decrypt_l0rdix_c2.py showing identified L0rdix traffic.

```
[+] Searching for screenshots...
[+] Dumped L0rdix screenshots in current directory (9):

c83ecd30-420b-4b88-aff4-36f1163a8fe4.jpg
1f70fa47-f2db-499d-854f-381f54981c93.jpg
6eaf3abc-6005-485e-97e4-13cd46930b04.jpg
649822d3-68e6-4830-9225-2222a234f6bf.jpg
4bc70d57-16aa-4e70-b354-5ca12fe9ae17.jpg
4321eae2-212d-469c-902b-b5e5325ac691.jpg
27420c0b-9bff-45f9-994b-68796ab90dab.jpg
03138ca0-1cf7-4bb0-96d2-120c882a9671.jpg
4d5a8631-3b4e-4b47-8541-6c1093ddd19a.jpg

[+] Decrypting strings using operator key (UTF-8): 3sc3RLrpd17
[+] AES key (hex): c57f4d4bcf08dcb7300e8bca8e5d3e22cedbaaee73bec7b027561975a825f54e
[+] IV (hex): 00000000000000000000000000000000
[+] Decrypted L0rdix C2 traffic (90 strings):

FCB25A42
Windows 7 Enterprise
Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz
Standard VGA Graphics Adapter

Admin
0
Main
1 GB
error
FCB25A42
Windows 7 Enterprise
Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz
Standard VGA Graphics Adapter
```

Figure 5 – Output from decrypt_l0rdix_c2.py showing extracted screenshots and decrypted traffic. You can see the decrypted values of L0rdix's configuration.

Source: https://www.bromium.com/decrypting-l0rdix-rats-c2/