An Analysis of L0rdix RAT, Panel and Builder | HP Wolf Security

By Alex Holland

Published: 2019-07-19 · Archived: 2026-04-05 19:20:58 UTC

L0rdix is a multipurpose remote access tool (RAT) that was first discovered being sold on underground criminal forums in November 2018. Shortly after its discovery, Ben Hunter of enSilo analysed the RAT's functionality. Although L0rdix's author set the price of the RAT at 4000 RUB (64 USD), for many cyber criminals even this was too high a price. In June 2019, a cracked version of the RAT's builder and admin panel began circulating through underground forums. I was especially curious in the admin panel to see if an analysis of it would lead to a better understanding of L0rdix and potentially improve its detection in the wild.

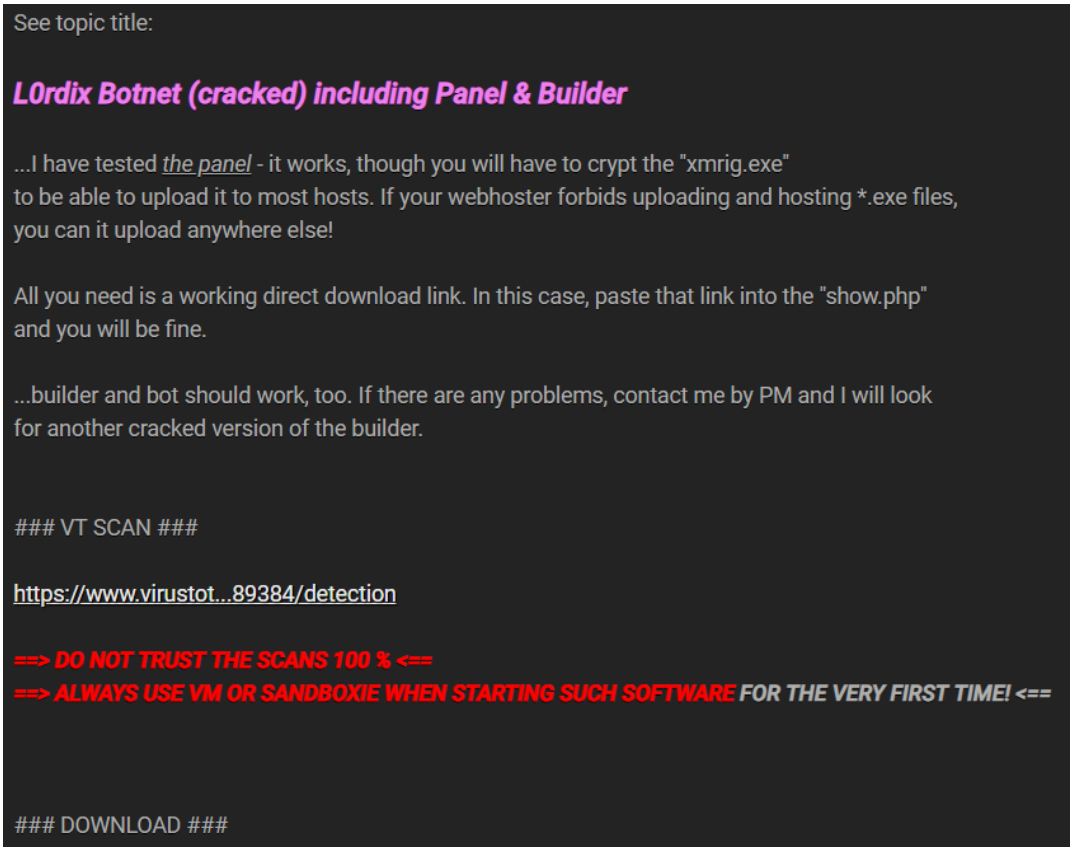

Figure 1 – Advert for a cracked copy of the L0rdix RAT panel and builder on an underground forum in June 2019.

L0rdix's Admin Panel

The admin panel consists of three components: a HTTP web server for the operator to administer their bots, a pre-made MySQL database for storing data from infected systems, and PHP scripts to send bot commands, process data received from bots and interface with the database.

By default, the URI of the L0rdix panel login page is webserver.tld/*admin_login*. Unlike many other RATs, L0rdix's login page is simple and does not advertise its namesake.

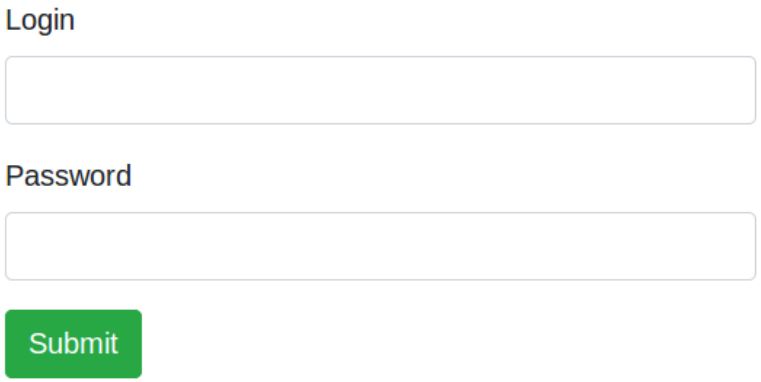

Figure 2 – Login page of L0rdix's panel.

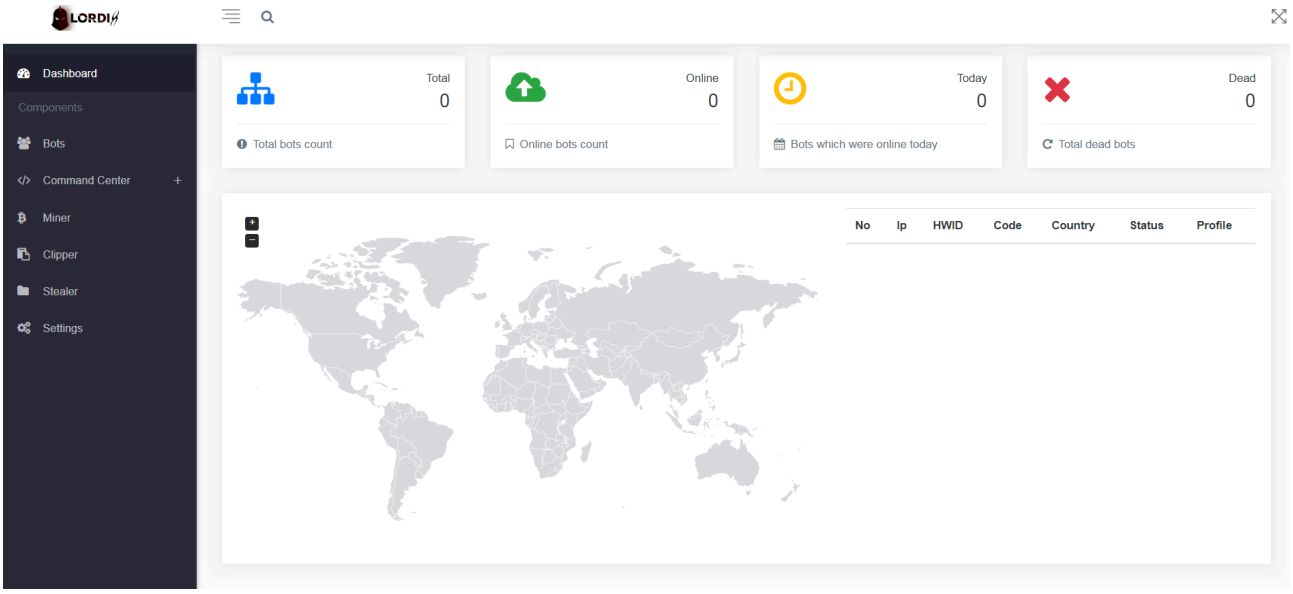

Figure 3 – The main dashboard of the L0rdix panel.

By querying the panel's MySQL database it was possible to understand the types of data L0rdix steals from its victims, its default configuration settings, and make an assessment about the sophistication of the malware. In the case of L0rdix, its database contains seven tables shown in figure 4, indicating that this RAT is not particularly complex.

```
mysql> SHOW TABLES;
+------------------+
| Tables_in_l0rdix |
+------------------+
| clips            |
| commands         |
| commands_done    |
| config           |
| logs             |
| victims          |
| wallets          |
+------------------+
7 rows in set (0.00 sec)
```

Figure 4 – Tables in the MySQL database for the L0rdix panel.

The “victims” table contains basic hardware and configuration information about infected systems, their location calculated using a geoIP database bundled with the panel, and the hash rate of the cryptominer (the open source Monero miner, XMRig) that L0rdix can command bots to download once persistent. Many of the fields relate to cryptomining, for instance one field is designated for the model of GPU used by the infected host, highlighting how important (and lucrative) L0rdix’s author considers cryptojacking as a monetisation activity. We’ve written about cryptojacking before on this blog. In our assessment, it’s likely that the rebound in the value of cryptocurrencies in the first half of 2019 is one of the drivers for the increase in cryptominer campaigns.

Default Panel Credentials

The “config” table contains the default login credentials to access the panel:

- Username: “root”
- Password: “toor”

It’s possible that L0rdix’s author is familiar with the Kali Linux distribution, given that they share the same default credentials.

```
mysql> SELECT * FROM config;
+-------+----------+-------------+---------------+------------------+--------------------+---------------+----------------+--------------+-----------+------------+
| login | password | start_delay | startup_delay | recover_interval | reconnect_interval | mining_status | warn_processes | warn_windows | exit_type | extensions |
+-------+----------+-------------+---------------+------------------+--------------------+---------------+----------------+--------------+-----------+------------+
| root  | toor     | 5           | 5             | 5                | 5                  | 1             |                |              | 1         | txt        |
+-------+----------+-------------+---------------+------------------+--------------------+---------------+----------------+--------------+-----------+------------+
1 row in set (0.00 sec)
```

Figure 5 – The default configuration of L0rdix RAT.

Out of the box, L0rdix operators are able to send eight commands to bots, although custom commands can be defined and added. These include:

- Download and execute
- Update
- Open page (visible)
- Open page (invisible)
- Cmd

- Kill process
- Upload file
- HTTP Flood

L0rdix's Command and Control (C2) Encryption and Decryption

L0rdix's C2 traffic is encrypted using AES symmetric-key encryption using a 256-bit key in Cipher Block Chaining (CBC) mode. When a sample is generated using L0rdix's builder the operator is able to decide the key to encrypt the C2 traffic. A SHA-256 hash is calculated from the operator's key. The first 32 characters (i.e. 256 bits) of the hashed operator key is used as the AES key in the encryption function. The panel's encryption function is implemented using the openssl_encrypt PHP function. The function requires a 16-byte initialisation vector (IV), but L0rdix's author decided to use 16 null bytes. The copy of the panel analysed contained a possible default operator key *3sc3RLrpd17*. The ciphertext is then Base64 encoded, with any plus (+) characters replaced with tildes (~) using PHP's str_replace function. L0rdix's decryption function is simply the reverse of the encryption function, using str_replace, base64_decode and openssl_decrypt.

Based on the implementation of the encryption and decryption functions it appears that each L0rdix panel operator must use the *same key for every bot they control*. From a detection standpoint, this is useful because L0rdix samples can be tied to specific actors based on common keys. Additionally, since the same key must be used for each panel, this means that it is possible to decrypt captured C2 traffic from any bots controlled by a panel where the key is known, for example where the key is extracted from a sample.

```
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 6 – L0rdix's server-side decryption function.

```
function encrypt($decrypted){
    $method = 'aes-256-cbc';

    $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);


    $encrypted = base64_encode(openssl_encrypt($decrypted, $method, $password, OPENSSL_RAW_DATA
    , $iv));

    $encrypted = str_replace("+", "~", $encrypted);
    return $encrypted;
}
```

Figure 7 – L0rdix's server-side encryption function.

L0rdix's Builder

The L0rdix builder is simple with only a few configurable options, including specifying the remote IP address or domain where the panel is hosted, generating a mutex to prevent the RAT from repeatedly re-infecting systems, and options to enable certain capabilities in the malware.

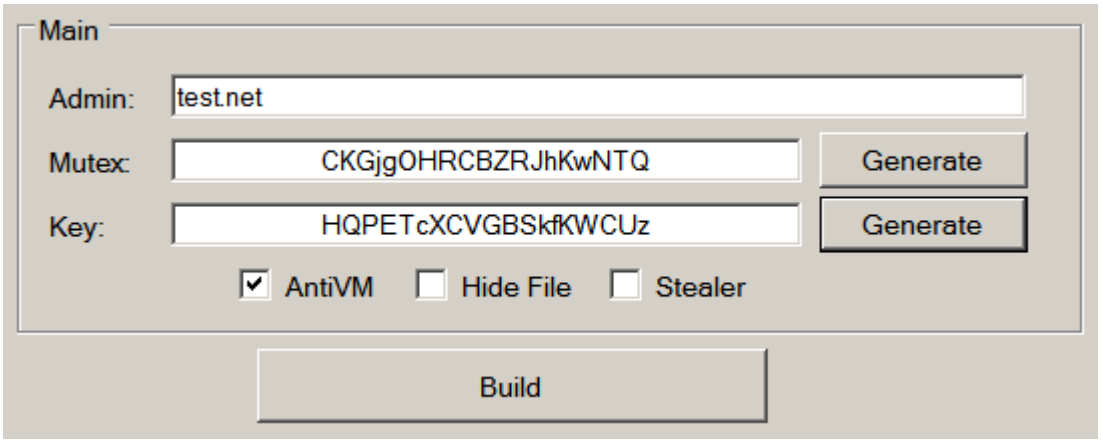

Figure 8 – L0rdix RAT builder.

L0rdix implements anti-analysis measures, such as enumerating running processes and stopping the RAT from exposing its functionality if common analysis tools are detected. The RAT also checks if it is running in a virtual machine or Sandboxie, a sandbox product.

```
if (Process.GetProcessesByName("PROCMON").Length != 0)
{
    return true;
}
if (Process.GetProcessesByName("NETSTAT").Length != 0)
{
    return true;
}
if (Process.GetProcessesByName("FILEMON").Length != 0)
{
    return true;
}
if (Process.GetProcessesByName("REGMON").Length != 0)
{
    return true;
}
if (Process.GetProcessesByName("NETMON").Length != 0)
{
    return true;
}
if (Process.GetProcessesByName("WIRESHARK").Length != 0)
{
    return true;
}
```

Figure 9 – L0rdix checks its runtime environment for common analysis tools.

The L0rdix bot tries to make its C2 communications blend in with legitimate traffic by using the User-Agent string for Firefox 53 on Windows 10, which is hardcoded in the RAT:

- Mozilla/5.0 (Windows NT 10.0; WOW64; rv:53.0) Gecko/20100101 Firefox/53.0

It also uses the WMI namespace, root\SecurityCenter2, to identify which antivirus is installed on the system, which is subsequently sent to the admin panel.

```
foreach (ManagementBaseObject managementBaseObject in new ManagementObjectSearcher(string.Format("\\\\{0}\\root\\SecurityCenter2",
  Environment.MachineName), "SELECT * FROM AntivirusProduct").Get())
{
    text = ((ManagementObject)managementBaseObject)["displayName"].ToString();
}
```

Figure 10 – L0rdix uses WMI to identify antivirus software installed on the infected system.

Conclusion

Based on this leak, L0rdix RAT has not evolved much since it first entered the scene in November 2018. Despite being advertised as a general purpose RAT, much of its functionality is geared towards cryptojacking. It is possible that this is a response by L0rdix's author to meet the rising demand for cryptomining botnets.

YARA Signature

As part of this research a YARA rule was written to detect L0rdix binaries, which we are sharing with the community.

```
rule win_l0rdix {
        meta:
                author = "Alex Holland (Bromium Labs)"
                date = "2019-07-19"
                sample_1 = "18C6AAF76985404A276466D73A89AC5B1652F8E9659473F5D6D656CA2705B0D3"
                sample_2 = "C2A4D706D713937F47951D4E6E975754C137159DC2C30715D03331FC515AE4E8"

        strings:
                $ua = "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:53.0) Gecko/20100101 Firefox/53.0" wi
                $sig = "L0rdix" wide ascii
                $sched_task = "ApplicationUpdateCallback" wide
                $exe = "syscall.exe" wide
                $cnc_url_1 = "connect.php?" wide
                $cnc_url_2 = "show.php" wide
                $browser_1 = "\\Kometa\\User Data\\Default\\Cookies" wide
                $browser_2 = "\\Orbitum\\User Data\\Default\\Cookies" wide
                $browser_3 = "\\Amigo\\User\\User Data\\Default\\Cookies" wide
                $coin_regex_1 = "[13][a-km-zA-HJ-NP-Z1-9]{25,34}" wide // Bitcoin
                $coin_regex_2 = "0x[a-fA-F0-9]{40}" wide // Ethereum
                $coin_regex_3 = "L[a-zA-Z0-9]{26,33}" wide // Litecoin

        condition:
```

```
                uint16(0) == 0x5A4D and (any of ($ua,$sig,$sched_task,$exe)) and (any of ($cnc_url_*
}
```

Source: https://www.bromium.com/an-analysis-of-l0rdix-rat-panel-and-builder/