MiKey - A Linux keylogger

Dec 14, 2016

Summary:

Linux malware is slowly becoming more popular. Within the past couple years there were several major incidents that cited the use of Windows backdoors being ported to Linux. Through our research on the Windows KLRD keylogger from the Odinaff report, we were able to discover several new keyloggers. The focus of this blog post is MiKey, a little-known and poorly detected keylogger.

At the time of this writing, the malware wasn't detected by a single engine on Virustotal.

SHA256:	9c07ed03f5bf56495e1d365552f5c9e74bb586ec45dffced2a8368490da4c829
File name:	mikey
Detection ratio:	0 / 54

Analysis

The malware is a 64 bit Linux executable:

```
9c07ed03f5bf56495e1d365552f5c9e74bb586ec45dffced2a8368490da4c829: ELF 64-b
```

And depends on the following libraries:

```
        linux-vdso.so.1 (0x00007ffd25123000)
        libX11.so.6 => /usr/lib/x86_64-linux-gnu/libX11.so.6 (0x00007f7f5642
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f7f5621c000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f7f55e7e000)
        libxcb.so.1 => /usr/lib/x86_64-linux-gnu/libxcb.so.1 (0x00007f7f55c5
        /lib64/ld-linux-x86-64.so.2 (0x00005597839c6000)
        libXau.so.6 => /usr/lib/x86_64-linux-gnu/libXau.so.6 (0x00007f7f55a5
        libXdmcp.so.6 => /usr/lib/x86_64-linux-gnu/libXdmcp.so.6 (0x00007f7f
```

Analyzing the symbol table for this binary yielded some interesting function names. (Full output omitted for readability):

```
    63: 00000000004014b2    79 FUNC    GLOBAL DEFAULT   14 createProccess
    64: 0000000000400ed6   128 FUNC    GLOBAL DEFAULT   14 initPlugins
    67: 0000000000400f56   105 FUNC    GLOBAL DEFAULT   14 moduleFeed
    68: 000000000040102d  1157 FUNC    GLOBAL DEFAULT   14 keylogger
    75: 0000000000400dc6   159 FUNC    GLOBAL DEFAULT   14 handleArgs
    83: 0000000000400e65   113 FUNC    GLOBAL DEFAULT   14 moduleHandleArgs
    85: 00000000004015fc   209 FUNC    GLOBAL DEFAULT   14 addData
    87: 0000000000400cd0    42 FUNC    GLOBAL DEFAULT   14 _start
    88: 0000000000400fbf   110 FUNC    GLOBAL DEFAULT   14 addParentheses
    92: 0000000000401501   126 FUNC    GLOBAL DEFAULT   14 main
   103: 0000000000400b00     0 FUNC    GLOBAL DEFAULT   11 _init
```

Comments left by the compiler provide evidence it was compiled on Ubuntu 16.04.2:

```
9c07ed03f5bf56495e1d365552f5c9e74bb586ec45dffced2a8368490da4c829:     file
```

Contents of section .comment:

```
 0000 4743433a 20285562 756e7475 20352e34  GCC: (Ubuntu 5.4
0010 2e302d36 7562756e 7475317e 31362e30  .0-6ubuntu1~16.0
0020 342e3229 20352e34 2e302032 30313630  4.2) 5.4.0 20160
0030 36303900                             609.
```

This is further evidenced by the build path in the binary:

```
/home/ubuntu/MiKey-64-ubuntu
```

The strace tool was used to quickly identify high-level function workflows and identify potential focus areas. One anomaly identified was a failed file opening, “mikey-text.so.” So we began there.

```
open("./tls/x86_64/mikey-text.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No suc
open("./tls/mikey-text.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file o
open("./x86_64/mikey-text.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such fil
open("./mikey-text.so", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or di
```

The malware isn’t explicitly searching these directories for mikey-text.so. This is a side effect of dlopen. From the man page:

```
“If, at the time that the program was started, the environment variable LD_
```

After a little searching we located a second binary (SHA-256 bc6d25dff00dfb68b19b362c409d2cf497e5dd97d9d6e5ce2bde2ba706f2bdb3) which contained the string “mikey-text.c.” From this, we assessed that mickey-text.so is the compiled version of this binary.

Using this assertion, we renamed the second binary to mikey-text.so and placed it in the load path identified using strace. This caused the successful execution of the malware. The output file (out.log) contained the logged keystrokes with associated timestamps.

```
[2016-12-13 14:14:29]
[2016-12-13 14:14:35]   (Enter)
[2016-12-13 14:14:49]  hello (space) from (space) a (space) keylogger
```

Through static analysis, we were able to identify that when the keylogger starts up, it loaded plugins, handle arguments, and then forked its process.

When loading the plugins, the keylogger looked for a single hardcoded plugin name “mikey-text.so” and called dlopen to obtain a handle to it.

```
loc_400EEC:
mov     eax, [rbp+i]
cdqe
lea     rdx, ds:0[rax*8]
mov     rax, [rbp+hdlarr]
lea     rbx, [rdx+rax]
mov     eax, [rbp+i]
cdqe
mov     rax, plugins[rax*8]
mov     esi, 2          ; mode
mov     rdi, rax
call    _dlopen
mov     [rbx], rax
mov     eax, [rbp+i]
cdqe
lea     rdx, ds:0[rax*8]
mov     rax, [rbp+hdlarr]
add     rax, rdx
mov     rax, [rax]
test    rax, rax
jnz     short loc_400F44
```

```
nop
add     rsp, 28h
pop     rbx
pop     rbp
retn
initPlugins endp
```

```
                public plugins
; const char *plugins[1]
plugins         dq offset aMikeyText_so ; DATA XREF: initPlugins+
_data           ends                    ; "mikey-text.so"
```

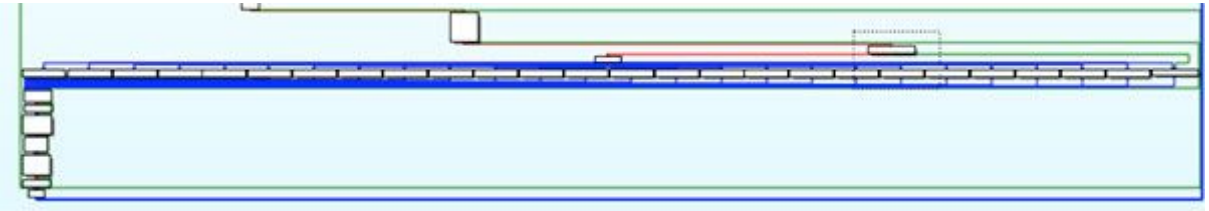
Once everything was loaded, the main functionality of the program was handled through the “keylogger” function.

To better understand Linux keyloggers and associated function calls, basic X function knowledge is critical. As a quick primer, here are some routines used by the “keylogger” function to query information about keystrokes or simply harvest raw keystroke data.

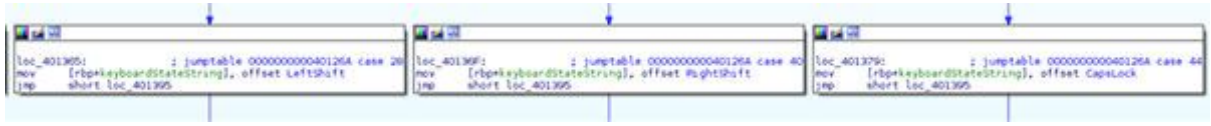
Function	Purpose
XOpenDisplay	Returns a display structure that serves as the connection to the X server. Communication is then carried out through TCP or IPC.

Function	Purpose
XQueryKeymap	Uses the structure from the display to gather information about the state of the keyboard. Information about which keys are currently pressed can be gathered using this call.
XkbKeycodeToKeysym	Uses the structure from the display to return the keysym for a particular key.
XKeysymToString	Converts the previously obtained keysym.
XGetInputFocus	Controls focus on the desktop.

Once the keycode is retrieved, it's compared against a large switch table to convert each keycode into a string. This is no different than most keyloggers.



Non-printable keystrokes are then identified and substituted with human-readable outputs.



If there is a non-printable character returned, a small method to format the string in parentheses is called to make for nice output into the log.

```
mov     rcx, rax
mov     edx, offset format ; " (%s) "
mov     esi, 0          ; maxlen
mov     edi, 0          ; s
mov     eax, 0
call    _snprintf
add     eax, 1
```

Once completed, the data is stored into a buffer and passed to a loadable module. The Linux dlsym method provides similar functionality as “LoadLibrary” on Windows. The previous handle from dlopen is being passed to dlsym, which we can now use to call the method “getFeed” from mikey-text.so.

```
mov     eax, [rbp+i]
cdqe
lea     rdx, ds:0[rax*8]
mov     rax, [rbp+hdlarr]
add     rax, rdx
mov     rax, [rax]
mov     esi, offset aGetfeed ; "getFeed"
mov     rdi, rax             ; handle
call    _dlsym
mov     [rbp+func], rax
call    _dlerror
mov     [rbp+result], rax
mov     rax, [rbp+b]
mov     rdx, [rbp+func]
mov     rdi, rax
mov     eax, 0
call    rdx
add     [rbp+i], 1
```

Peering into the “getFeed” function on mikey-text.so it simply calls the _log function.

```
push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+b], rdi
mov     rax, [rbp+b]
mov     rsi, rax
lea     rdi, aS             ; " %s\n"
mov     eax, 0
call    _log
nop
leave
retn
```

The _log function will call _time and _localtime (to harvest the timestamps) and build these into a format string.

```
push    rdx
mov     ecx, edi
mov     edx, esi
lea     rsi, format     ; "[%04d-%02d-%02d %02d:%02d:%02d] "
mov     rdi, rax        ; s
mov     eax, 0
call    _sprintf
```

At this point, the output file is opened for writing with the appended (“a+”) flag and the file is written to using the _fputs method. If no option for –output was provided to mikey-text.so then the default name of “out.log” is

provided. The screenshot below identifies the contents of cs:outputfile_ptr as a pointer to the name of the output file.

```
mov     rax, cs:outputfile_ptr
mov     rax, [rax]
lea     rsi, modes          ; "a+"
mov     rdi, rax            ; filename
call    _fopen
mov     [rbp+file], rax
cmp     [rbp+file], 0
jz      short loc_C03
mov     rdx, [rbp+file]
lea     rax, [rbp+logbuf]
mov     rsi, rdx            ; stream
mov     rdi, rax            ; s
call    _fputs
mov     rax, [rbp+file]
mov     rdi, rax            ; stream
call    _fclose
```

Outside of a small method to parse arguments, there isn't much more functionality to mikey-text.so. It's a simple logging plugin for the main MiKey keylogger. Booz Allen assesses that additional plugins may exist (for C2 communication, or to hook to other files), but is unable to confirm at this time.

To run the keylogger and give a custom argument for an output file named keylogged.txt, the following command can be used. In addition, providing the “-b” option will “background” the process.

```
~/Desktop# ./mikey_keylogger -b --output keylogged.txt
```

Checking processes on the host, the command “ps aux” was issued.

```
root      6332  0.1  0.1  17992  2032 pts/1    S    15:29   0:00 ./mikey_keylogger -b --output keylogged.txt
```

And checking the output of the keylogged file:

```
Open ▾                                   keylogged.txt
                                         ~/Desktop
[2016-12-13 15:29:47]
[2016-12-13 15:29:53]   (Enter)
[2016-12-13 15:30:04]  ps (space) aux
[2016-12-13 15:30:20]   (unknown)
[2016-12-13 15:31:13]   (Enter)
```

Conclusion

Small utilities that are built for a specific purpose often bypass AV with ease. Attackers are able to write a functional keylogger that will dump the contents to a local file. By having modular code, the authors could build plugins that achieve whatever task they need. The plugin nature of this code also puts the reverse engineer at a disadvantage. Without access to each module, only specific known functions of the tool can be documented.

One unnerving aspect of this keylogger is that, without an active command and control capability, the attacker would need to be confident in their ability to repeatedly gain remote access to the victim computer to retrieve the keylogged information.

All it takes to catch this is basic process and file monitoring, but if our Linux field experience is any indicator, there aren't many shops with this level of visibility on non-Windows workstations.

This project is maintained by securitykitten