

The KLRD Keylogger

Published: 2016-11-28 · Archived: 2026-04-05 16:26:42 UTC

Symantec released a [report in the beginning of October that talks about Odinaff](#), which is a new piece of malware used in campaigns targeting financial institutions. In the report, Symantec posts several of the auxiliary tools used in the campaign and many of the associated droppers. Booz Allen Intelligence Analysts wanted to take a closer look at some of these binaries and post some analysis so that network defenders can better understand how these tools work. In some cases the simple advice of “Install AV” is enough, but all too often is insufficient when looking at small targeted utilities.

At the time of this writing, a little under 40% of AV engines are detecting the keylogger utility (21/56 on Virustotal.com).

```
SHA256:      e07267bbfcbff72a9aff1872603ffbb630997c36a1d9a565843cb59bc5d97d90
File name:   klr.exe
Detection ratio: 21 / 56
```

Following the naming convention of klr.exe, the output file of the logged keystrokes is named klr.log (which is stored in the C:\Windows\Temp\ directory). The keylogger has no exfil capability, so one unnerving aspect of writing to a local log is that the attacker has access to the host via some other means. If a tool like this is discovered on your network, you need to check the compromised host for malicious lateral connections or backdoor connections.

The malware is also compiled with a build path of:

```
d:\Programming\C++\projects\klr\bin\klr.pdb
```

Keylogging

This program is very straightforward in its execution. The first thing that it does is start the keylogging thread.

```
push    eax                ; lpThreadId
push    esi                ; dwCreationFlags
push    esi                ; lpParameter
push    offset StartAddress ; lpStartAddress
push    esi                ; dwStackSize
push    esi                ; lpThreadAttributes
mov     [ebp+ThreadId], esi
call    ds:CreateThread
```

The thread obtains a handle to the current process by calling `GetModuleHandleA`

```
call    memset
add     esp, 0Ch
push   ebx                ; lpModuleName
call   ds:GetModuleHandleA
cmp    eax, ebx
jnz   short loc_4014BF
```

Then it does some quick error checking to make sure that it could obtain a handle. If it fails, it tosses an error and tries to call `LoadLibraryA` on it. If both fail, the keylogger exits.

Assuming success, the keylogger sets a hook by calling `SetWindowsHookEx`. The hook procedure is followed next.

```
loc_4014BF:                ; dwThreadId
push   ebx
push   eax                ; hmod
push   offset KeyEvent@12 ; lpfn
push   0Dh                ; idHook
call   ds:SetWindowsHookExA
...
```

Following `_KeyEvent@12` in a debugger, the hook procedure performs some simple bounds checking on an obtained key and then tries to obtain the current window's (foreground window) text and the thread process.

```
call  dword ptr ds:[<&GetWindowText>]
test  eax, eax
jne   keylogger_klrd.401253
call  dword ptr ds:[<&GetLastError>]
push  eax
push  keylogger_klrd.40200C
call  keylogger_klrd.401000
pop   ecx
pop   ecx
jmp   keylogger_klrd.401266
lea   eax, dword ptr ss:[ebp-118]
push  eax
push  keylogger_klrd.4020FC
call  keylogger_klrd.401000
pop   ecx
pop   ecx
mov   eax, dword ptr ss:[ebp-350]
mov   dword ptr ds:[403000], eax
push  0
push  dword ptr ss:[ebp-350]
call  dword ptr ds:[<&GetWindowThreadProcessId>]
mov   dword ptr ss:[ebp-138], eax
cmp   dword ptr ss:[ebp-138], 0
jne   keylogger_klrd.4012A3
call  dword ptr ds:[<&GetLastError>]
push  eax
push  keylogger_klrd.402104
call  keylogger_klrd.401000
pop   eax
```

```
40200C: "#2 Can't get window text, #kd\r\n"
4020FC: "\r\n%ks:"
402104: "Can't get thread id, #kd\r\n"
```

If this information cannot be obtained, the output log contains error messages of “Can’t get window text” and “Can’t get thread id”.

A small lookup table is provided to check the keystroke against known control characters

```
mov byte ptr ss:[ebp-354],a1
cmp byte ptr ss:[ebp-354],8
je keylogger_klrd.401336
cmp byte ptr ss:[ebp-354],D
je keylogger_klrd.401356
cmp byte ptr ss:[ebp-354],10
je keylogger_klrd.401373
cmp byte ptr ss:[ebp-354],14
je keylogger_klrd.401380
cmp byte ptr ss:[ebp-354],1B
je keylogger_klrd.401346
cmp byte ptr ss:[ebp-354],20
je keylogger_klrd.401366
jmp keylogger_klrd.40138D
push keylogger_klrd.402120
call <keylogger_klrd.writeToLog>
pop ecx
jmp keylogger_klrd.4013E6
push keylogger_klrd.402128
call <keylogger_klrd.writeToLog>
pop ecx
jmp keylogger_klrd.4013E6
push keylogger_klrd.402130
call <keylogger_klrd.writeToLog>
pop ecx
jmp keylogger_klrd.4013E6
push keylogger_klrd.40213C
call <keylogger_klrd.writeToLog>
pop ecx
jmp keylogger_klrd.4013E6
push keylogger_klrd.402144
call <keylogger_klrd.writeToLog>
pop ecx
jmp keylogger_klrd.4013E6
push keylogger_klrd.402150
call <keylogger_klrd.writeToLog>
pop ecx
```

	402120:" <8kSp>"
	402128:" <Esc>"
	402130:" <Enter>"
	40213C:" <Sp>"
	402144:" <Shift>"
	402150:" <Caps>"

And if the key is not in the lookup table (switch table), the default case occurs and it is converted to Ascii and written to the log file. This switch statement is very similar to the MSDN provided code for using keyboard input.

```
case WM_CHAR:
    switch (wParam)
    {
        case 0x08:
            // Process a backspace.
            break;
        case 0x0A:
            // Process a linefeed.
            break;
        case 0x1B:
            // Process an escape.
            break;
        case 0x09:
            // Process a tab
            break;
        case 0x0D:
            // Process a carriage return.
            break;
        default:
```

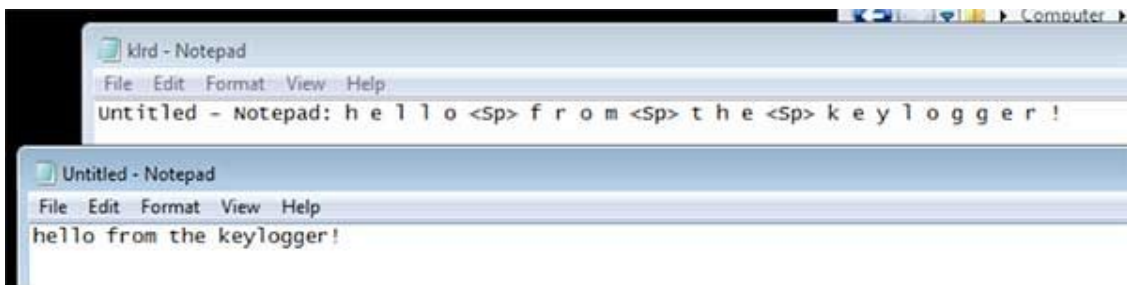
```
        // Process displayable characters.  
        break;  
    }  
}
```

The method for writing the log file is interesting because rather than keeping a handle open to the file and just writing to it whenever possible, the function gets a handle each time and opens it to write each individual character.

The path and name for the log file is hard-coded as C:\Windows\Temp\klrd.log

```
push     esi             ; hTemplateFile  
push     esi             ; dwFlagsAndAttributes  
push     4               ; dwCreationDisposition  
push     esi             ; lpSecurityAttributes  
push     esi             ; dwShareMode  
push     0C0000000h      ; dwDesiredAccess  
push     offset FileName ; "C:\\Windows\\Temp\\klrd.log"  
call     ds:CreateFileA
```

At this point a majority of the keylogger is documented. Running it provides an output log that looks like the following.



While not the most sophisticated keylogger, its basic functionality is effective and allows the binary to be only 5kb in size, all while avoiding detection from over 60% of AV engines.

Source: <https://securitykitten.github.io/2016/11/28/the-klrd-keylogger.html>