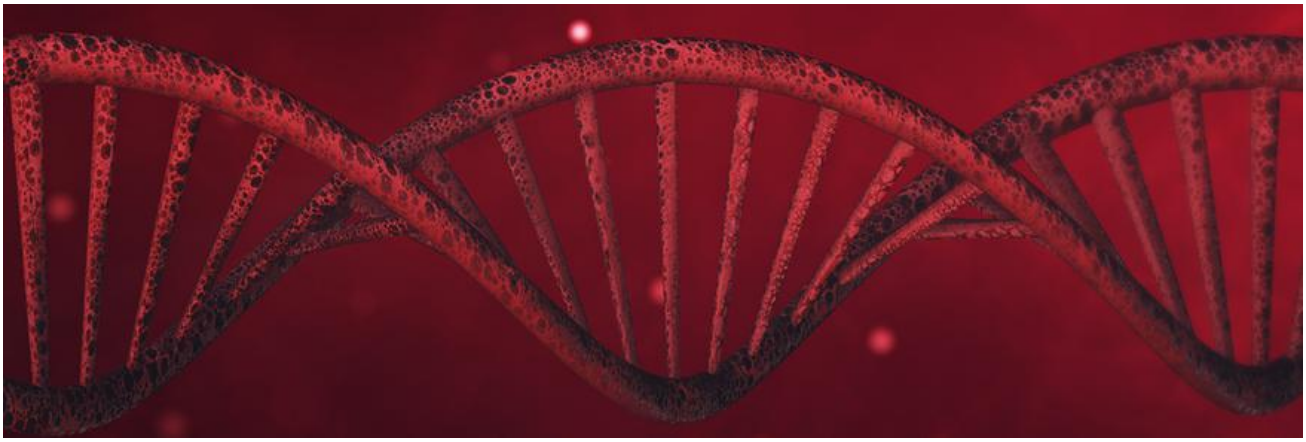
Threat Spotlight: Virlock Polymorphic Ransomware

blogs.blackberry.com/en/2019/07/threat-spotlight-virlock-polymorphic-ransomware

The BlackBerry Cylance Threat Research Team

RESEARCH & INTELLIGENCE / 07.15.19 / The BlackBerry Cylance Threat Research Team



Virlock is a polymorphic, file-infecting ransomware first discovered in 2014. In 2016 it demonstrated new capabilities allowing it to spread through shared applications and cloud storage. When executed, it drops three instances of itself. One instance carries out the file infection, another locks the machine, and the third creates a persistence mechanism by registering as a service. Attackers demand bitcoin payment from victims who want their systems unlocked.

The polymorphic nature of Virlock means every instance is different from a heuristic perspective, a tactic that effectively bypasses signature-based antivirus (AV) solutions. For example, Virlock drops three instances of itself during the first stage of its attack, each one implementing different obfuscation and persistence mechanisms. By varying the functionality that each instance implements, Virlock guarantees all three instances can evade a signature-based detection system.

Technical Analysis

Virlock decrypts and runs the original lure file when executing the first time on a non-infected machine. When executing on a previously infected machine it checks to see if a ransom has been paid. On successfully ransomed machines Virlock will decrypt the host file but take no further malicious actions. If a machine has not been successfully ransomed Virlock executes a malicious behavior (Figure 1) without decrypting the host file.

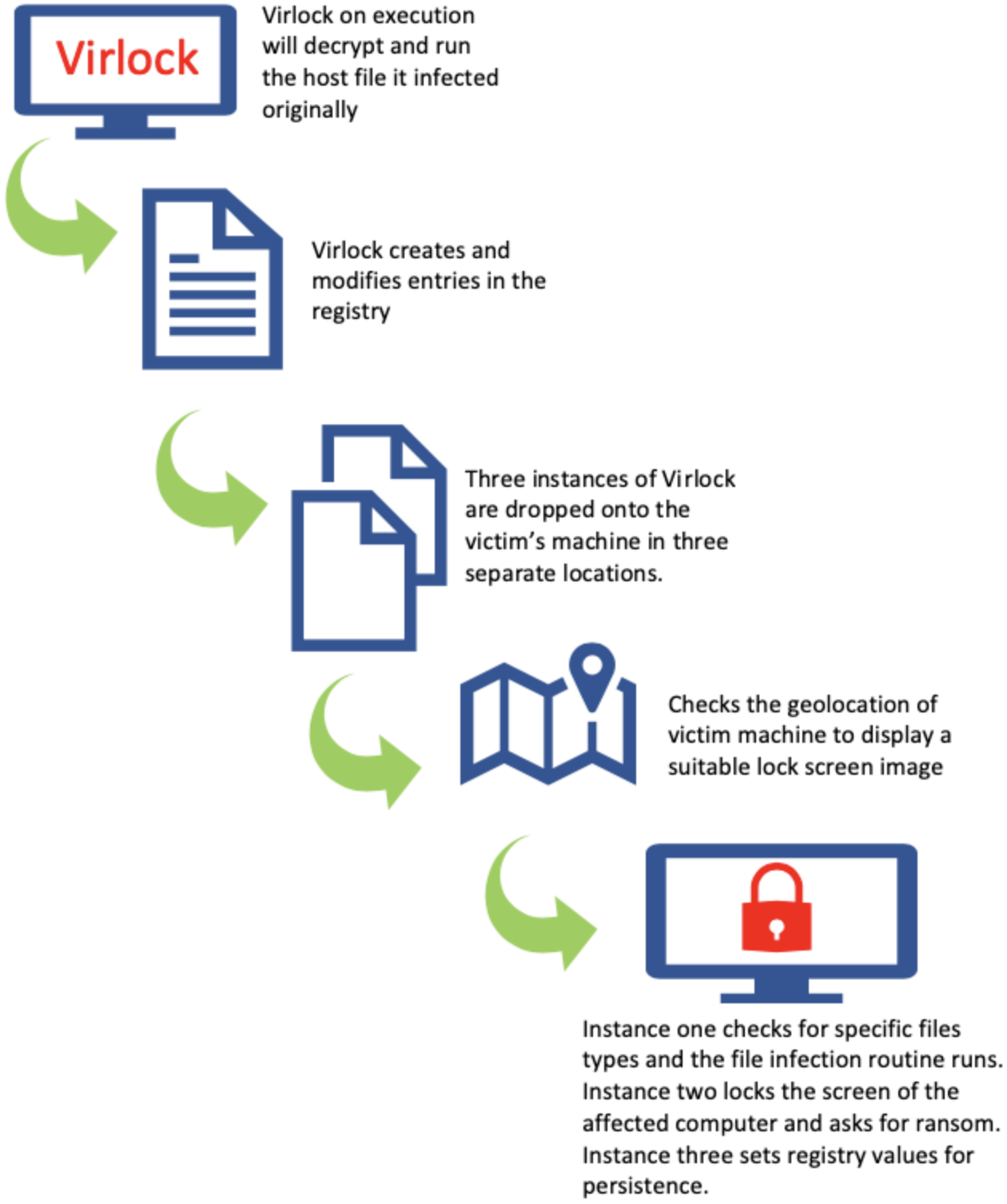


Figure 1: Execution flow

The malicious file was heavily obfuscated and included anti-debugging techniques that complicated analysis. During our investigation we observed the decoding and launch of the second stage payload:

```
.text:004A16A8 decode_run_second_stage proc near       ; CODE XREF: j_decode_run_second_stage↑j
.text:004A16A8                 sub     esp, 3B4h
.text:004A16AE                 mov     esi, offset second_stage
.text:004A16B3                 jmp     loc_4A16CE
```

Figure 2: Jump to decode_run_second_stage

The decryption routine uses an incremental xor loop to decrypt the second stage. It checks to verify that the file has been fully decrypted, then calls the second stage:

```
.text:004A16B8
.text:004A16B8 decrypt_loop:                           ; CODE XREF: decode_run_second_stage+48↓j
.text:004A16B8                 mov     [edi], al
.text:004A16BA                 nop
.text:004A16BB                 inc     edx
.text:004A16BC                 inc     esi
.text:004A16BD                 inc     edi
.text:004A16BE                 nop
.text:004A16BF                 dec     ecx
.text:004A16C0                 cmp     ecx, 0
.text:004A16C3                 jnz     next_byte
.text:004A16C9                 jmp     call_second_stage
.text:004A16CE ; ---------------------------------------------------------------------------
.text:004A16CE
.text:004A16CE loc_4A16CE:                             ; CODE XREF: decode_run_second_stage+B↑j
.text:004A16CE                 mov     edi, offset buffer
.text:004A16D3                 mov     [edi], eax      ; allocated memory ptr
.text:004A16D5                 mov     edi, eax
.text:004A16D7                 nop
.text:004A16D8                 mov     ebx, edi
.text:004A16DA                 nop
.text:004A16DB                 mov     ecx, 948        ; second stage size
.text:004A16E0                 jmp     $+5
.text:004A16E5 ; ---------------------------------------------------------------------------
.text:004A16E5
.text:004A16E5 loc_4A16E5:                             ; CODE XREF: decode_run_second_stage+38↑j
.text:004A16E5                 mov     edx, 8
.text:004A16EA
.text:004A16EA next_byte:                              ; CODE XREF: decode_run_second_stage+1B↑j
.text:004A16EA                 mov     al, [esi]
.text:004A16EC                 nop
.text:004A16ED                 xor     al, dl          ; incremental XOR starting from 8
.text:004A16EF                 nop
.text:004A16F0                 jmp     decrypt_loop
.text:004A16F5 ; ---------------------------------------------------------------------------
.text:004A16F5
.text:004A16F5 call_second_stage:                      ; CODE XREF: decode_run_second_stage+21↑j
.text:004A16F5                 call    ebx
.text:004A16F7                 add     esp, 3B4h
.text:004A16FD                 retn
.text:004A16FD decode_run_second_stage endp
.text:004A16FD
```

Figure 3: Decryption loop and call to second stage

Next, the second stage decrypts the data representing the instructions to be executed, in a virtualized way, by the third stage stub function. Once decrypted, the instructions bytes are injected into the “execute_opcode” function between RVA 0x1032 and 0x103B as shown in Figure 4.

```
seg000:00001000 execute_opcode  proc near
seg000:00001000
seg000:00001000 arg_4           = dword ptr  8
seg000:00001000 arg_8           = dword ptr  0Ch
seg000:00001000
seg000:00001000                 mov     [esp+arg_4], ebp
seg000:00001004                 nop
seg000:00001005                 mov     ebp, esp
seg000:00001007                 cmp     [ebp+arg_8], 0
seg000:0000100B                 jz      short loc_1010
seg000:0000100D                 mov     esp, [ebp+arg_8]
seg000:00001010
seg000:00001010 loc_1010:                               ; CODE XREF: execute_opcode+B↑j
seg000:00001010                 nop
seg000:00001011                 add     ebp, 0C003E7B5h
seg000:00001017                 add     ebp, 3FFC1ADBh
seg000:0000101D                 add     esp, 0C003F111h
seg000:00001023                 add     esp, 3FFC0FDFh
seg000:00001029                 pop     esi
seg000:0000102A                 pop     edi
seg000:0000102B                 pop     edx
seg000:0000102C                 nop
seg000:0000102D                 pop     ecx
seg000:0000102E                 nop
seg000:0000102F                 pop     ebx
seg000:00001030                 pop     eax
seg000:00001031                 popf
seg000:00001032                 inc     edx             ; virtual command (changes each time this function is called)
seg000:00001033                 nop                     ; ;
seg000:00001034                 nop                     ; ;
seg000:00001035                 nop                     ; ;
seg000:00001036                 nop                     ; ;
seg000:00001037                 nop                     ; ;
seg000:00001038                 nop                     ; ;
seg000:00001039                 nop                     ; ;
seg000:0000103A                 nop                     ; ;
seg000:0000103B                 nop
seg000:0000103C                 pushf
seg000:0000103D                 push    eax
seg000:0000103E                 nop
seg000:0000103F                 push    ebx
seg000:00001040                 push    ecx
seg000:00001041                 push    edx
```

Figure 4: "inc edx" followed by a sequence of NOPs injected into the stub function

Payload Execution

The ransomware drops three instances of itself in three different locations upon execution:

Instance one:

File path: %AllUsersProfile%\<Random_folder_name>\<Random_file_ name>.exe

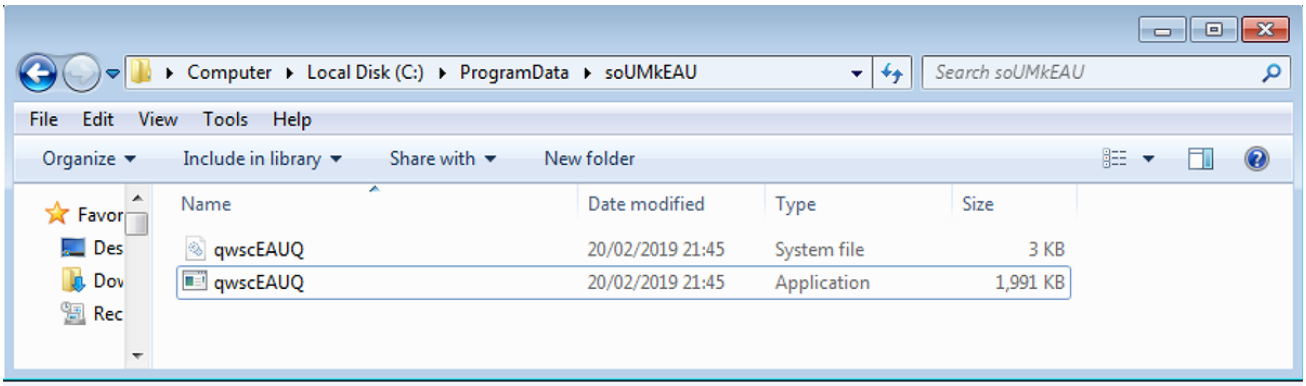


Figure 5: First instance dropped by original file

Instance two:

File path: %UserProfile%\<Random_folder_name>\<Random_file_name>.exe

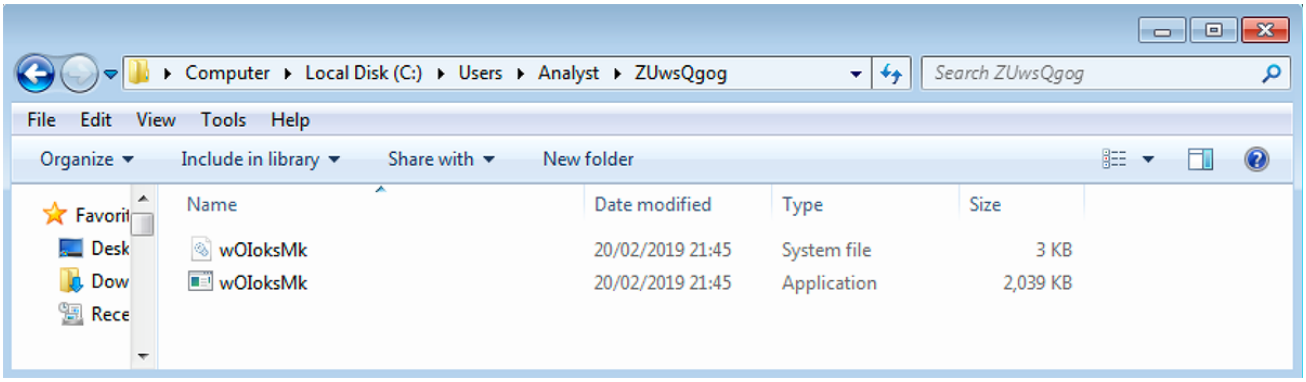


Figure 6: Second instance dropped by original file

Instance three:

File path: %AllUsersProfile%\<Random_folder_name>\<Random_file_ name>.exe

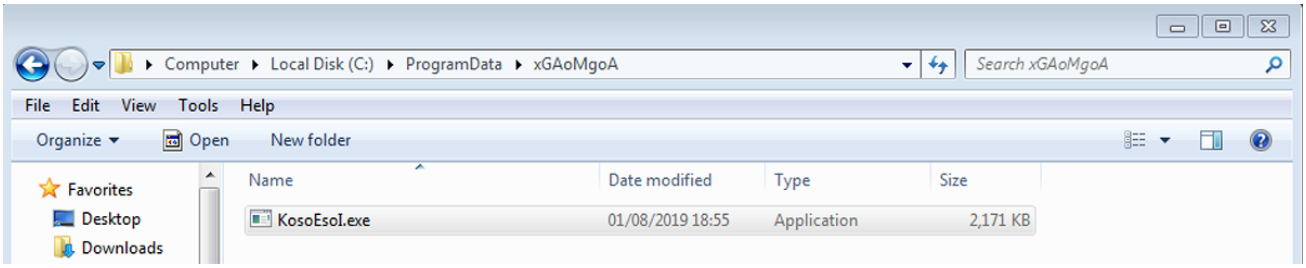


Figure 7: Third instance dropped by original file

Persistence

Virlock modifies the system registry to create persistence on an infected system. It sets two registry values of two randomly named instances of itself in the "**CurrentVersion\Run"** registry key. These settings cause the programs to run each time a user logs on:

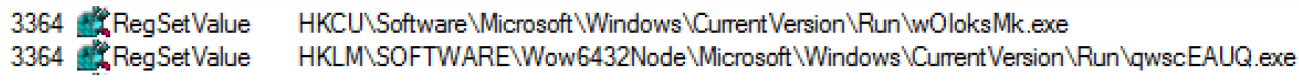


Figure 8: Persistence via Run Registry Key modifications

Virlock modifies a registry key that specifies which program WinLogon executes when a user logs on:

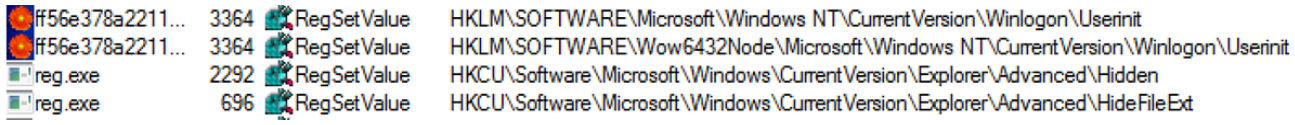


Figure 9: Persistence via Userinit Registry Value modification

Virlock's registry key modification:

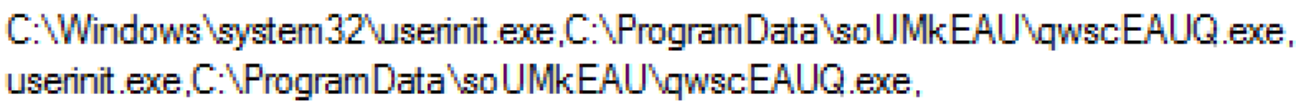


Figure 10: Registry modification for persistence

The third instance sets a value for a randomly named service:

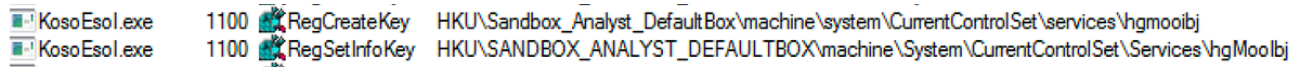


Figure 11: Persistence via new service registration

File Infection

After dropping three instances of itself, Virlock checks for specific file types to infect. Once the targeted file have been identified, the malware encrypts the file and then replaces it with a copy of the virus code with the encrypted original file content appended to it.

Targeted File Types:

.exe	.doc	.xls	.pdf	.ppt	.mdb
.zip	.rar	.mp3	.mpg	.wma	.png
.gif	.bmp	.jpg	.jpeg	.psd	.p12
.cer	.crt	.p7b	.pfx	.pem	

Virlock appends an .exe file extension to all infected files, then modifies the registry to hide the file extensions. This is done to trick the user into executing the infected host file:

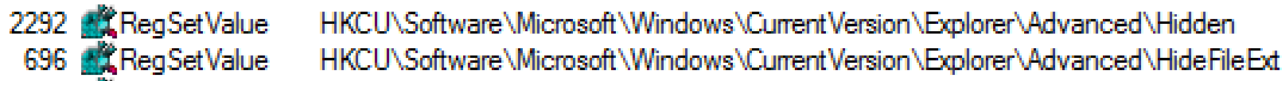


Figure 12: File extension registry change

Here is an example of the Notepad++ executable before and after infection:

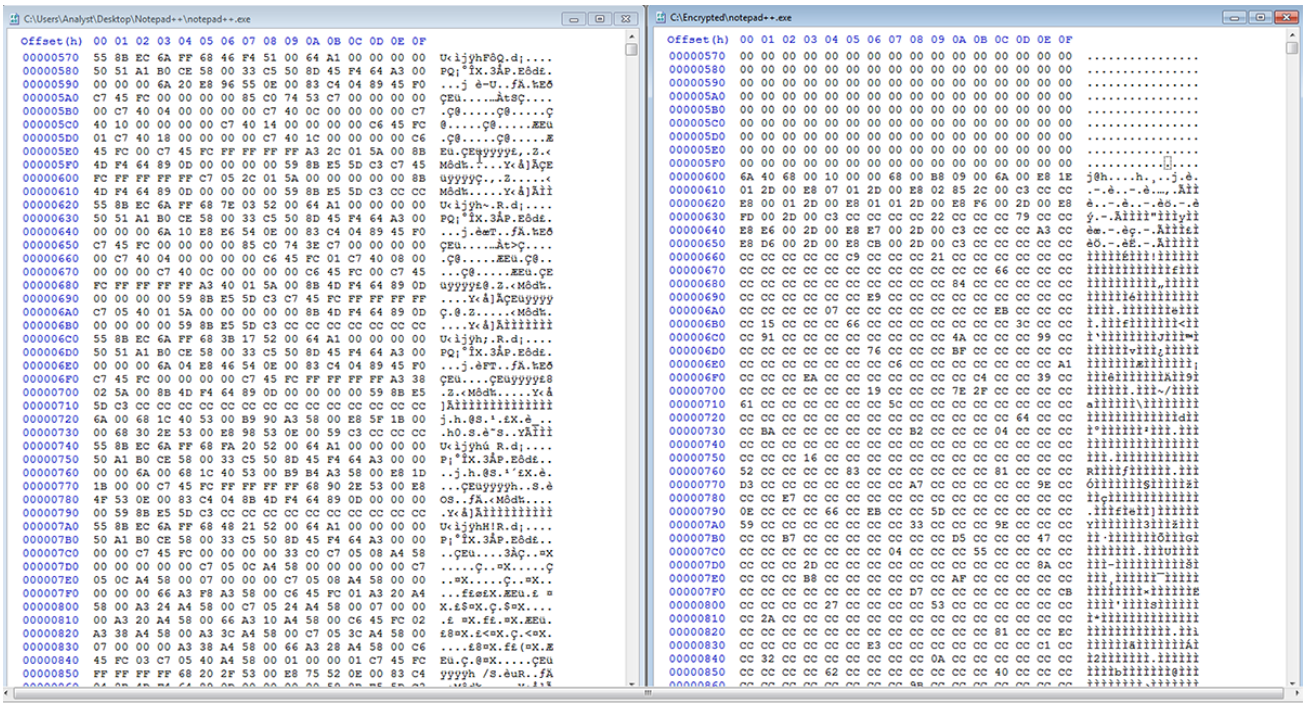


Figure 13: Before and after infection

Locked Screen

While Virlock is running and executing its file infection routine, its second instance locks the screen of the victim's machine. During this process Virlock also shuts down the process explorer.exe and task manager. It checks the geolocation of the device by searching the registry and displays a message tailored to the victim's location. The screen-lock message asks victims to pay a ransom in bitcoin:

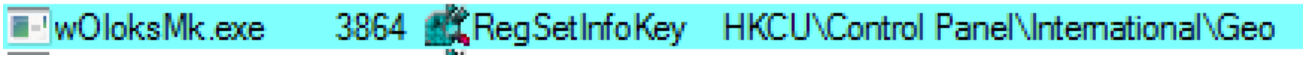


Figure 14: Checks registry geolocation of the machine

Figure 15: Lock screen message

Why Virlock is Important and Why You Should be Concerned

Virlock was first detected in 2014 but made resurgent appearances in 2016 and 2017. With each reappearance Virlock demonstrated new capabilities, indicating that the malware is actively developed and updated by cyber criminals. Virlock deploys an impressive triple-instance attack strategy and a location-specific ransom screen threatening users with fake legal action should they refuse to comply. With ransomware attacks costing organizations roughly $13,000 USD per incident, Virlock is a threat that businesses cannot afford to ignore.

BlackBerry Cylance Stops Virlock

BlackBerry Cylance uses artificial intelligence (AI)-based agents trained for threat detection on millions of both safe and unsafe files. This lets BlackBerry Cylance spot a threat based on countless file attributes instead of a specific file signature. Virlock is a polymorphic threat, capable of modifying its code, but this evasive tactic does not fool our AI-driven security agents. Our solutions analyze and convict threats based upon millions of threat

features, not specific file signatures. Blackberry Cylance, which offers a predictive advantage over zero-day threats, is trained on and effective against both new and legacy cyber threatsattacks.

(EDITOR'S NOTE: This blog was updated on 8/5/2019 to add in extra detail from an extended technical analysis performed by our Threat Research team).

About The BlackBerry Cylance Threat Research Team

The BlackBerry Cylance Threat Research team examines malware and suspected malware to better identify its abilities, function and attack vectors. Threat Research is on the frontline of information security and often deeply examines malicious software, which puts us in a unique position to discuss never-seen-before threats.

Back