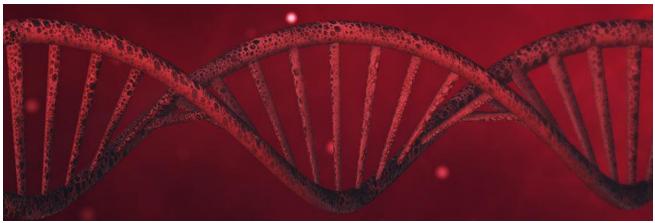
Threat Spotlight: Virlock Polymorphic Ransomware

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 <u>original lure file</u> when executing the first time on a noninfected 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 <u>executes a malicious behavior</u> (Figure 1) without decrypting the host file.

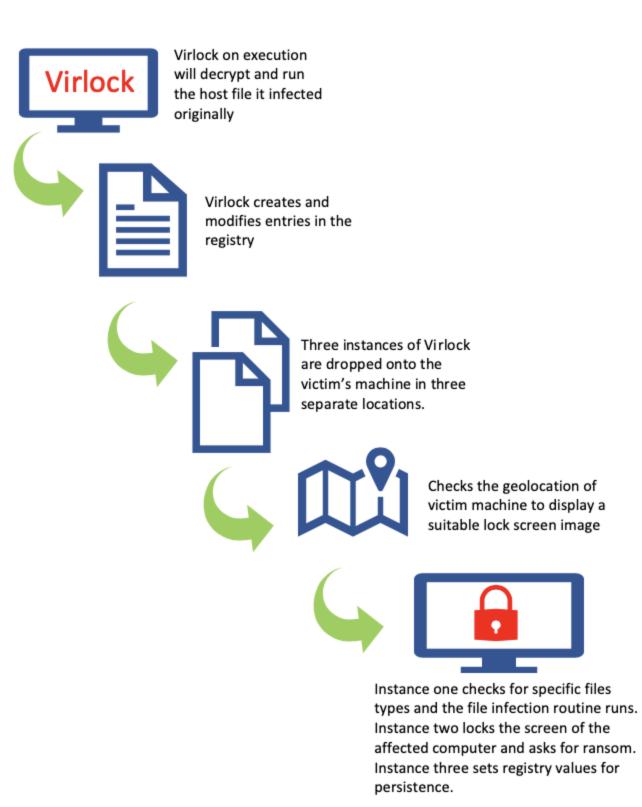


Figure 1: Execution flow

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:004A16/ | No N8 decode_run_second_stage | e proc near | ; CODE XREF: j_decode_run_second_stage^j |
|---|---------------|----------------------------------|-------------|--|
| • | .text:004A16 | NS sub | esp, 3B4h | |
| | .text:004A16 | | esi, offset | second_stage |
| Г | .text:004A16 | 33 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:

| <pre>text:004A1668 decrypt_loop:</pre> | | .text:004A16B8 | - | | | | |
|---|------------------|-----------------|------------------|---------|---------------|------|---|
| <pre>text:004A1688 mov [edi], al text:004A1688 inc edx text:004A168B inc edi text:004A168B inc edi text:004A168F dec ecx text:004A1660 cmp ecx, 0 text:004A16C5 j nc edi.stage text:004A16C5 i second_stage text:004A16C5 mov edi, offset buffer text:004A1605 mov edi, eax text:004A1605 mov ecx, 948 ; second stage size text:004A1665 mov edx, 8 text:004A1665 i second_stage text:004A1665 mov edx, 8 text:004A1665 mov e</pre> | | .text:004A16B8 | decrypt loop: | | | | CODE XREF: decode run second stage+48↓i |
| <pre>text:004A16BA nop text:004A16BC inc edx .text:004A16BC inc esi .text:004A16BC inc esi .text:004A16BF dec ecx .text:004A16BF dec ecx .text:004A16C3 jnz next_byte .text:004A16C3 jnz next_byte .text:004A16C5 jnz next_byte .text:004A16C5 inc edi, offset buffer .text:004A16CE text:004A16CE mov edi, offset buffer .text:004A16C5 mov edi, eax ; allocated memory ptr .text:004A16D5 mov edi, eax .text:004A16D5 mov ecx, 948 ; second stage size .text:004A16D5 mov ecx, 948 ; second stage size .text:004A16E5 inc editer inc edi .text:004A16E5 mov edx, 84 ; second stage size .text:004A16E5 mov edx, 8 .text:004A16E5 mov e</pre> | | | | nov | [edi], al | 1 | , |
| <pre>.text:004A16BB inc edx .text:004A16BC inc esi .text:004A16BC inc edi .text:004A16BF dec ecx .text:004A16CB cmp ecx, 0 .text:004A16CB cmp ecx, 0 .text:004A16CB cmp ecx, 0 .text:004A16CC imp ecx, 0 .text:004A16CE imp ecx, 0 .text:004A16CE imp ecx, 0 .text:004A16CB cmov edi, offset buffer .text:004A16D3 mov [edi], eax ; allocated memory ptr .text:004A16D3 mov edx, edi .text:004A16CB imp ext.out edi .text:004A16CB imp ecx, 948 ; second stage size .text:004A16E5 imp ecx, 948 ; coDE XREF: decode_run_second_stage+Btj .text:004A16E5 imp ecx, 948 ; coDE XREF: decode_run_second_stage+Btj .text:004A16E5 imp ecx, 8 .text:004A16E5 imp ecx, 948 ; imp ext, 944 imp ext, 944 imp ecx, 944 imp ecx, 948 ; i</pre> | • | | | | [], | | |
| <pre>text:004A16BC inc esi text:004A16BC inc edi text:004A16BF dec ecx text:004A16BF dec ecx text:004A16C3 jnz next_byte text:004A16C3 jnz next_byte text:004A16C5 ; text:004A16CE ; text:004A16C5 mov edi, offset buffer text:004A16D5 mov edi, eax text:004A16BB mov ebx, edi text:004A16BB mov ecx, 948 ; second stage size text:004A16E5 ; text:004A16E5 inc_4A16E5 inc</pre> | • | | | | edx | | |
| <pre>.text:004A16BD inc edi .text:004A16BF dec ecx .text:004A16C6 cmp ecx, 0 .text:004A16C3 jnz next_byte .text:004A16C3 jnz next_byte .text:004A16C5 ; .text:004A16C5 cmov edi, offset buffer .text:004A16C5 mov edi, ffset buffer .text:004A16D3 mov [edi], eax ; allocated memory ptr .text:004A16D5 mov edi, eax .text:004A16D6 mov ebx, edi .text:004A16D8 mov ecx, 948 ; second stage size .text:004A16D8 mov ecx, 948 ; second stage size .text:004A16E5 ; .text:004A16E5 ; .text:004A16E5 mov edx, 8 .text:004A16E5 mov edx, 8 .text:004A16E5 mov edx, 8 .text:004A16E5 mov al, [esi] .text:004A16E5 mov al,</pre> | • | | | | | | |
| <pre>.text:004A16BE nop text:004A16CF dec ecx .text:004A16C3 jnz next_byte .text:004A16C3 jnz next_byte .text:004A16C5 j .text:004A16CE ; .text:004A16CE i .text:004A16CE i .text:004A16CE i .text:004A16C5 mov edi, offset buffer .text:004A16D5 mov edi, eax ; allocated memory ptr .text:004A16D5 mov ebx, edi .text:004A16D8 mov ebx, edi .text:004A16D8 mov ecx, 948 ; second stage size .text:004A16E5 ; .text:004A16E5 i .text:004A16E5 i .text:004A16E5</pre> | • | .text:004A16BD | | inc | edi | | |
| <pre>.text:004A16BF dec ecx 0 .text:004A16CB cmp ecx, 0 .text:004A16C3 jnz next_byte .text:004A16C5 jmp call_second_stage .text:004A16CE ; .text:004A16CE cloc_4A16CE: .text:004A16C5 mov edi, offset buffer .text:004A16D3 mov [edi], eax ; allocated memory ptr .text:004A16D5 mov edi, eax .text:004A16D5 mov ecx, 948 ; second stage size .text:004A16E0 mov ecx, 948 ; second stage size .text:004A16E5 ; .text:004A16E5 inc_4A16E5: .text:004A16E5 inc_4A16E5 inc_4A16E5: .text:004A16E5 inc_4A16E5 inc_4E</pre> | • | .text:004A16BE | | nop | | | |
| <pre>text:004A16C3 jnz next_byte text:004A16C6 j text:004A16CE i text:004A16CE loc_4A16CE: text:004A16CE loc_4A16CE: text:004A16CB loc_4A16CE text:004A16D3 mov edi, eax ; allocated memory ptr text:004A16D5 mov edi, eax text:004A16D8 mov ebx, edi text:004A16D8 mov ecx, 948 ; second stage size text:004A16E5 j text:004A16E5 i text:004A16E5 i text:004A16E5 i text:004A16E5 i text:004A16E5 i text:004A16E5 mov edx, 8 text:004A16E5 mov edx, 8 text:004A16E5 mov edx, 8 text:004A16E5 mov edx, 8 text:004A16E5 i text:004A16E5 mov edx, 8 text:004A16E5 mov edx, 8 te</pre> | • | | | | ecx | | |
| <pre>text:004A16C3 jnz next_byte text:004A16C6 j text:004A16CE i text:004A16CE loc_4A16CE: text:004A16C1 loc_4A16CE: text:004A16C3 mov edi, offset buffer text:004A16D3 mov [edi], eax ; allocated memory ptr text:004A16D5 mov edi, eax text:004A16D8 mov ebx, edi text:004A16D8 mov ecx, 948 ; second stage size text:004A16E5 j text:004A16E5 inc_4A16E5: text:004A16E5 inc_4A16E5: text:004A16E5 inc_4A16E5: text:004A16E5 mov edx, 8 text:004A16E5 inc_4A16E5 mov edx, 8 text:004A16E5 nop text:004A16E6 next_byte: text:004A16E5 nop text:004A16E5 nop text:004A16E5 nop text:004A16E5 nop text:004A16E5 inc_4A16E nop text:004A16E5 nop text:004A16E5 inc_4A16E nop text:004A16E5 inc_4A16E nop text:004A16E5 nop text:004A16E5 inc_4A16E nop text:004A16E5 inc_4A16E nop text:004A16E5 inc_4A16E nop text:004A16E nop text:004A16E5 inc_4A16E nop text:004A16E5 inc_4A16E nop text:004A16E5 inc_4A16E nop text:004A16E nop text:004</pre> | • | .text:004A16C0 | | cmp | ecx, 0 | | |
| <pre>text:004A16CG jmp call_second_stage .text:004A16CE ; .text:004A16CE loc_4A16CE: ; CODE XREF: decode_run_second_stage+B†j .text:004A16DS mov edi, offset buffer .text:004A16DS mov edi, eax ; allocated memory ptr .text:004A16DS mov edi, eax ; .text:004A16DB mov ebx, edi .text:004A16DB mov ecx, 948 ; second stage size .text:004A16E5 ; .text:004A16E5 ; .text:004A16E5 ; .text:004A16E5 ; .text:004A16E5 ; .text:004A16E5 mov edx, 8 .text:004A16E5 ic_4A16E5 ; .text:004A16E5 mov edx, 8 .text:004A16E5 ic_4A16E5 ; .text:004A16E5 mov edx, 8 .text:004A16E5 ic_4A16E5 ; .text:004A16E5 mov edx, 8 .text:004A16E5 mov al, [esi] incremental XOR starting from 8 .text:004A16E5 ; .text:004A16E5 mov al, al, al i incremental XOR starting from 8 .text:004A16E5 ; .text:004A16E5 imp decrypt_loop itext:004A16E5 ; .text:004A16E5 ; .text:004A16E5 imp decrypt_loop itext:004A16E5 ; .text:004A16E5 imp decrypt_loop itext:004A16E5 ; .text:004A16E5 ; .text:004A16E5 imp decrypt_loop itext:004A16E5 ; .text:004A16E5 imp decrypt_loop itext:004A16E5 ; .text:004A16E5 imp decrypt_loop itext:004A16E5 imp decrypt_loop itext:004A16E5 ; .text:004A16E5 imp decrypt_loop itext:004A16E5 imp decrypt_loop itext:00</pre> | | .text:004A16C3 | | | next byte | | |
| <pre>.text:004A16CE ; .text:004A16CE loc_4A16CE: ; CODE XREF: decode_run_second_stage+B†j .text:004A16D3 mov edi, eax ; allocated memory ptr .text:004A16D5 mov edi, eax .text:004A16D7 nop .text:004A16D8 mov ebx, edi .text:004A16D8 mov ecx, 948 ; second stage size .text:004A16E5 mov ecx, 948 ; second stage size .text:004A16E5 ;</pre> | | .text:004A16C9 | | | | tage | |
| <pre>.text:004A16CE .text:004A16CE loc_4A16CE: .text:004A16CE loc_4A16CE: .text:004A16DS mov edi, offset buffer .text:004A16D3 mov [edi], eax ; allocated memory ptr .text:004A16D5 mov edi, eax .text:004A16D8 mov ebx, edi .text:004A16D8 mov ebx, edi .text:004A16D8 mov ecx, 948 ; second stage size .text:004A16E0 jmp \$+5 .text:004A16E5 ; .text:004A16E5 mov edx, 8 .text:004A16E5 mov al, [esi] .text:004A16EA mov al, [esi] .text:004A16EF nop .text:004A16EF nop .text:004A16EF mov edx, 8 .text:004A16EF mov edx, 8 .text:004A16EC mov al, [esi] .text:004A16EF mov al, [esi] .text:004A16EF mov edx, 8 .text:004A16EF mov al, [esi] .text:004A16EF mov al, [esi] .text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+18†j .text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21†j .text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21†j .text:004A16F7 add esp, 3B4h .text:004A16F7 metn .text:004A16F7 metn</pre> | | .text:004A16CE | | | | | |
| <pre>.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:004A16D8 mov ecx, 948 ; second stage size .text:004A16E5 ; .text:004A16E5 ; .text:004A16E5 ioc_4A16E5: .text:004A16E5 mov edx, 8 .text:004A16E5 mov edx, 8 .text:004A16E5 mov edx, 8 .text:004A16EA next_byte: .text:004A16EA next_byte: .text:004A16EA next_byte: .text:004A16EF nop .text:004A16EF nop .text:004A16EF nop .text:004A16EF s; .text:004A16EF nop .text:004A16EF s; .text:004A16EF nop .text:004A16F5 s; .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16FD retn .text:004A16FD retn .text:004A16FD retn .text:004A16FD retn</pre> | i i | .text:004A16CE | - | | | | |
| <pre>.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:004A16D8 mov ecx, 948 ; second stage size .text:004A16E5 ; .text:004A16E5 ; .text:004A16E5 ioc_4A16E5: .text:004A16E5 mov edx, 8 .text:004A16E5 mov edx, 8 .text:004A16E5 mov edx, 8 .text:004A16EA next_byte: .text:004A16EA next_byte: .text:004A16EA next_byte: .text:004A16EF nop .text:004A16EF nop .text:004A16EF nop .text:004A16EF s; .text:004A16EF nop .text:004A16EF s; .text:004A16EF nop .text:004A16F5 s; .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16FD retn .text:004A16FD retn .text:004A16FD retn .text:004A16FD retn</pre> | | .text:004A16CE | loc 4A16CE: | | | ; | CODE XREF: decode run second stage+B^j |
| <pre>.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:004A16E5 j</pre> | _ i \ • | | - | mov | edi, offset b | | |
| <pre>.text:004A16D7 nop .text:004A16D8 mov ebx, edi .text:004A16D8 mov ecx, 948 ; second stage size .text:004A16E0 jmp \$+5 .text:004A16E5 ; .text:004A16E5 icc_4A16E5: ; CODE XREF: decode_run_second_stage+38†j .text:004A16E5 mov edx, 8 .text:004A16E5 mov edx, 8 .text:004A16EA next_byte: ; CODE XREF: decode_run_second_stage+18†j .text:004A16EA next_byte: ; CODE XREF: decode_run_second_stage+18†j .text:004A16EC nop .text:004A16ED xor al, dl ; incremental XOR starting from 8 .text:004A16F5 ;</pre> | • | .text:004A16D3 | 1 | mov | [edi], eax | ; | allocated memory ptr |
| <pre>.text:004A16D8 mov ebx, edi .text:004A16DA nop .text:004A16DB mov ecx, 948 ; second stage size .text:004A16E5 j jmp \$+5 .text:004A16E5 ;</pre> | i • | .text:004A16D5 | | mov | edi, eax | | |
| <pre>text:004A16D8 mov ebx, ed1 text:004A16DB mov ecx, 948 ; second stage size text:004A16E6 jmp \$+5 text:004A16E5 ; text:004A16E5 ioc_4A16E5: text:004A16E5 mov edx, 8 text:004A16E5 mov edx, 8 text:004A16EA mov al, [esi] text:004A16EA mov al, [esi] text:004A16EC nop text:004A16EC nop text:004A16EF nop text:004A16EF nop text:004A16F5 ; text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21†j text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21†j text:004A16FD retn text:004A16FD retn text:004A16FD retn</pre> | | .text:004A16D7 | | nop | | | |
| <pre>.text:004A16D8 mov ecx, 948 ; second stage size .text:004A16E6 jmp \$+5 .text:004A16E5 ;</pre> | • | .text:004A16D8 | 1 | mov | ebx, edi | | |
| <pre>.text:004A16E0 jmp \$+5 .text:004A16E5 ; .text:004A16E5 loc_4A16E5: ; CODE XREF: decode_run_second_stage+38†j .text:004A16E5 loc_4A16E5: ; CODE XREF: decode_run_second_stage+38†j .text:004A16EA mov edx, 8 .text:004A16EA next_byte: ; CODE XREF: decode_run_second_stage+18†j .text:004A16EA mov al, [esi] .text:004A16EC nop .text:004A16EF nop .text:004A16EF nop .text:004A16F5 ; .text:004A16F5 ; .text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21†j .text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21†j .text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21†j .text:004A16F7 add esp, 3B4h .text:004A16FD retn .text:004A16FD decode_run_second_stage endp</pre> | • | .text:004A16DA | | nop | | | |
| <pre>.text:004A16E5 ; .text:004A16E5 loc_4A16E5: .text:004A16E5 mov edx, 8 .text:004A16E5 mov edx, 8 .text:004A16EA next_byte: .text:004A16EA next_byte: .text:004A16EC nop .text:004A16ED xor al, dl ; incremental XOR starting from 8 .text:004A16FF nop .text:004A16F5 ; .text:004A16F5 ; .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16F7 add esp, 3B4h .text:004A16FD retn .text:004A16FD decode_run_second_stage endp</pre> | • | .text:004A16DB | 1 | mov | ecx, 948 | ; | second stage size |
| <pre>.text:004A16E5 .text:004A16E5 loc_4A16E5: .text:004A16E5 loc_4A16E5: .text:004A16E5 mov edx, 8 .text:004A16EA next_byte: .text:004A16EA next_byte: .text:004A16EA mov al, [esi] .text:004A16EC nop .text:004A16EF nop .text:004A16EF nop .text:004A16F5 ; .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage endp</pre> | | .text:004A16E0 | | jmp | \$+5 | | |
| <pre>.text:004A16E5 loc_4A16E5: .text:004A16E5 mov edx, 8 .text:004A16EA .text:004A16EA next_byte: .text:004A16EA next_byte: .text:004A16EA mov al, [esi] .text:004A16EC nop .text:004A16ED xor al, dl ; incremental XOR starting from 8 .text:004A16FF nop .text:004A16F5 ; .text:004A16F5 ; .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage endp</pre> | | .text:004A16E5 | ; | | | | |
| <pre>.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:004A16FF nop .text:004A16F5 ; .text:004A16F5 ; .text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21†j .text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21†j .text:004A16F7 add esp, 3B4h .text:004A16FD retn .text:004A16FD decode_run_second_stage endp</pre> | | .text:004A16E5 | | | | | |
| <pre>.text:004A16EA .text:004A16EA next_byte: .text:004A16EA next_byte: .text:004A16EA mov al, [esi] .text:004A16EC nop .text:004A16ED xor al, dl ; incremental XOR starting from 8 .text:004A16FF nop .text:004A16F5 ; .text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21^j .text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21^j .text:004A16F7 add esp, 3B4h .text:004A16FD retn .text:004A16FD decode_run_second_stage endp</pre> | | .text:004A16E5 | loc_4A16E5: | | | ; | CODE XREF: decode_run_second_stage+38^j |
| <pre>.text:004A16EA next_byte: .text:004A16EA mov al, [esi] .text:004A16EC nop .text:004A16ED xor al, dl ; incremental XOR starting from 8 .text:004A16EF nop .text:004A16F5 ; .text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21^j .text:004A16F5 call_second_stage: ; CODE XREF: decode_run_second_stage+21^j .text:004A16F7 add esp, 3B4h .text:004A16FD retn .text:004A16FD decode_run_second_stage endp</pre> | 1 4 | .text:004A16E5 | 1 | mov | edx, 8 | | |
| <pre>.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 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</pre> | | .text:004A16EA | | | | | |
| <pre>.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 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 retn .text:004A16FD decode_run_second_stage endp</pre> | | .text:004A16EA | next_byte: | | | ; | CODE XREF: decode_run_second_stage+1B^j |
| <pre>.text:004A16ED xor al, dl ; incremental XOR starting from 8 .text:004A16EF nop .text:004A16F0 jmp decrypt_loop .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</pre> | | .text:004A16EA | 1 | mov | al, [esi] | | |
| <pre>.text:004A16EF nop .text:004A16F0 jmp decrypt_loop .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</pre> | | .text:004A16EC | | nop | | | |
| <pre>.text:004A16F0 jmp decrypt_loop .text:004A16F5 ; .text:004A16F5</pre> | | .text:004A16ED | : | xor | al, dl | ; | incremental XOR starting from 8 |
| <pre>.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</pre> | | .text:004A16EF | | nop | | | |
| <pre>.text:004A16F5 .text:004A16F5 call_second_stage: .text:004A16F5 call_second_stage: .text:004A16F5 call_ebx .text:004A16F7 add esp, 3B4h .text:004A16FD retn .text:004A16FD decode_run_second_stage endp</pre> | | .text:004A16F0 | | jmp | decrypt_loop | | |
| <pre>.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</pre> | | | ; | | | | |
| <pre>.text:004A16F5 call ebx .text:004A16F7 add esp, 3B4h .text:004A16FD retn .text:004A16FD decode_run_second_stage endp</pre> | | | | | | | |
| .text:004A16F5 Call eDx .text:004A16F7 add esp, 3B4h .text:004A16FD retn .text:004A16FD decode_run_second_stage endp | | | | | | ; | CODE XREF: decode_run_second_stage+21↑j |
| <pre>.text:004A16FD retn .text:004A16FD decode_run_second_stage endp</pre> | | | | | | | |
| .text:004A16FD decode_run_second_stage endp | | | | | esp, 3B4h | | |
| | | | | | | | |
| I.text:004A16FD | | | decode_run_secon | d_stage | endp | | |
| | | J.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+Bij |
| 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

| | | | | | | × |
|----------------|--------------------------------|--------------------------------|--------------|-----------------|---|---|
| € • € | Computer Local Disk (C:) Pro | ogramData 🕨 soUMkEAU | • * j | Search soUMkEAU | J | ۶ |
| ile Edit Viev | v Tools Help | | | | | |
| Organize 🔻 | Include in library | New folder | | | | ? |
| 🔶 Favor | Name | Date modified | Туре | Size | | |
| ka Des | 🚳 qwscEAUQ | 20/02/2019 21:45 | System file | 3 KB | | |
| 鷆 Dov 📃 Rec | wscEAUQ | 20/02/2019 21:45 | Application | 1,991 KB | | |
| | | | | | | |

Figure 5: First instance dropped by original file

Instance two:

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

| • ال | Computer → Local Disk (C:) | ► Users ► Analyst ► ZUwsQgog | - - ↓ | Search ZUwsQgog | | - 0 | × |
|----------------|----------------------------|------------------------------|--------------|-----------------|-----|-----|---|
| File Edit View | / Tools Help | | | | | | |
| Organize 🔻 | Include in library 👻 Share | with 🔻 New folder | | | 800 | • | 2 |
| 숨 Favorit | Name | Date modified | Туре | Size | | | |
| 🧮 Desk | wOIoksMk | 20/02/2019 21:45 | System file | 3 KB | | | |
| 🚺 Dow | wOloksMk | 20/02/2019 21:45 | Application | 2,039 KB | | | |
| 🕮 Rece | | | | | | | |
| - | | | | | | | |

Figure 6: Second instance dropped by original file

Instance three:

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

| Cor Cor | nputer 🕨 Local Disk (C:) 🕨 ProgramData | ▶ xGAoMgoA | | ▼ 🍫 Search xGi | |
|--------------------------|--|------------------|-------------|----------------|-----------------|
| File Edit View To | ools Help Open Newfolder | | | | i≡ - - 0 |
| ☆ Favorites | A Name | Date modified | Туре | Size | |
| 💻 Desktop ᠾ Downloads | KosoEsoI.exe | 01/08/2019 18:55 | Application | 2,171 KB | |

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:

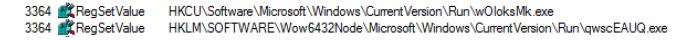


Figure 8: Persistence via Run Registry Key modifications

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:

C:\Windows\system32\userinit.exe,C:\ProgramData\soUMkEAU\qwscEAUQ.exe, userinit.exe,C:\ProgramData\soUMkEAU\qwscEAUQ.exe,

Figure 10: Registry modification for persistence

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

KosoEsol.exe
 1100
 RegCreateKey
 HKU\Sandbox_Analyst_DefaultBox\machine\system\CurrentControlSet\services\hgmooibj
 1100
 RegSetInfoKey
 HKU\SANDBOX_ANALYST_DEFAULTBOX\machine\System\CurrentControlSet\Services\hgmooibj

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:

2292 RegSetValue 696 RegSetValue

ue HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden ue HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\HideFileExt

Figure 12: File extension registry change

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

| 🟦 C:\Users\Analy | yst\Desktop\Notepad++\notepad++.exe | | C\Encrypted\notepad++.exe | - • • |
|------------------|--|----------|---|-------|
| Offset(h) | 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F | <u>^</u> | Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F | |
| 00000570 | 55 8B EC 6A FF 68 46 F4 51 00 64 A1 00 00 00 00 UkijyhFôg.d; | | 00000570 00 00 00 00 00 00 00 00 00 00 00 00 0 | |
| 00000580 | 50 51 A1 B0 CE 58 00 33 C5 50 8D 45 F4 64 A3 00 PQ: "IX. 3AP.E | | 00000580 00 00 00 00 00 00 00 00 00 00 00 00 0 | |
| 00000590 | 00 00 00 6A 20 E8 96 55 0E 00 83 C4 04 89 45 F0 j è-UfA | .teð | 00000590 00 00 00 00 00 00 00 00 00 00 00 00 0 | |
| 000005A0 | C7 45 FC 00 00 00 00 85 C0 74 53 C7 00 00 00 00 CEUÀtSC | | 000005A0 00 00 00 00 00 00 00 00 00 00 00 00 0 | |
| 000005B0 | 00 c7 40 04 00 00 00 c7 40 0c 00 00 00 c7 .cece. | ç | 000005B0 00 00 00 00 00 00 00 00 00 00 00 00 0 | |
| 000005c0 | 40 10 00 00 00 00 c7 40 14 00 00 00 00 c6 45 FC 0ç0 | .ÆEQ | 000005c0 00 00 00 00 00 00 00 00 00 00 00 00 0 | |
| 00000500 | 01 c7 40 18 00 00 00 c7 40 1c 00 00 00 c6 .ç9ç9 | ž | 000005D0 00 00 00 00 00 00 00 00 00 00 00 00 0 | |
| | | | 000005E0 00 00 00 00 00 00 00 00 00 00 00 00 0 | |
| 000005F0 | 4D F4 64 89 0D 00 00 00 00 59 8B E5 5D C3 C7 45 Môdt Yea |)ĂÇE | 000005F0 00 00 00 00 00 00 00 00 00 00 00 00 0 | |
| | | | 00000600 6A 40 68 00 10 00 00 68 00 B8 09 00 6A 00 E8 1E j@hh.,.j.e. | |
| | 4D F4 64 89 0D 00 00 00 00 59 8B E5 5D C3 CC CC Môdh Yea | | 00000610 01 2D 00 E8 07 01 2D 00 E8 02 85 2C 00 C3 CC CCè, ÅÌÌ | |
| | 55 8B EC 6A FF 68 7E 03 52 00 64 A1 00 00 00 00 Ukijyh~.R.d; | | 00000620 E8 00 01 2D 00 E8 01 01 2D 00 E8 F6 00 2D 00 E8 èèè | |
| | 50 51 A1 B0 CE 58 00 33 C5 50 8D 45 F4 64 A3 00 PQ; "ÎX.3ÅP.E | | 00000630 FD 00 2D 00 C3 CC CC CC C2 2C CC CC 79 CC CC ýAiiii"iiiyii | |
| | 00 00 00 6A 10 E8 E6 54 0E 00 83 C4 04 89 45 F0j.emTfA | | 00000640 E8 E6 00 2D 00 E8 E7 00 2D 00 C3 CC CC CC A3 CC èmècÄÌÌÌÈÌ | |
| | | | 00000650 E8 D6 00 2D 00 E8 CB 00 2D 00 C3 CC CC CC CC è6èÉÀÌÌÌÌÌ | |
| | | | 00000660 cc | |
| | 00 00 00 C7 40 0C 00 00 00 00 C6 45 FC 00 C7 45 Ç9 EE | | 00000670 cc | |
| | FC FF FF FF FF A3 40 01 5A 00 8B 4D F4 64 89 0D ugggge8.2. <m< td=""><td>ôd%.</td><td>00000680 cc cc</td><td></td></m<> | ôd%. | 00000680 cc | |
| | | | 00000690 cc | |
| | | | 000006A0 cc | |
| 00000680 | | | 000006B0 cc 15 cc cc cc 66 cc cc cc cc cc cc cc cc cc | |
| 000006C0 | | | 000006c0 cc 91 cc 99 cc i'ilililililili | |
| | | | 000006D0 CC CC CC CC CC CC CC CC BF CC CC CC CC 111111v111,11111 000006E0 CC A1 11111111 | |
| | | | | |
| | | | 000006F0 CC CC CC EA CC CC CC CC CC CC CC CC C3 9 CC 1116111111111191 00000700 CC CC CC CC CC CC 19 CC CC CC 7E 2F CC CC CC CC 111111.111-/111 | |
| 00000700 | | | 00000710 61 cc | |
| 00000710 | | | | |
| 00000720 | | | 00000720 CC | |
| | | | 00000740 CC | |
| | 55 8B EC 6A FF 68 FA 20 52 00 64 A1 00 00 00 00 U(1jyhú R.d; 50 A1 B0 CE 58 00 33 C5 50 8D 45 F4 64 A3 00 00 P:°IX.3ÅP.Eô | | | |
| | | | 00000760 52 cc cc cc cc 83 cc Rititfitititititit | |
| | 1B 00 00 C7 45 FC FF FF FF FF 68 90 2E 53 00 E8CEuyyyyh. | | 00000770 D3 CC 95 CC CC ATTINITIE | |
| | 4F 53 0E 00 83 C4 04 8B 4D F4 64 89 0D 00 00 00 0Sfä. <m8dt< td=""><td></td><td>00000780 cc cc</td><td></td></m8dt<> | | 00000780 cc | |
| | 00 59 8B E5 5D C3 CC | | 00000790 DE CC CC CC 66 CC EB CC CC 5D CC CC CC CC CC CC . lilffelililili | |
| | 55 8B EC 6A FF 68 48 21 52 00 64 A1 00 00 00 00 U(ljyhH!R.d; | | 000007A0 59 CC CC CC CC CC CC 33 CC CC CC 9E CC CC CC Y11111111 | |
| | 50 A1 B0 CE 58 00 33 C5 50 8D 45 F4 64 A3 00 00 P1°1X.3ÅP.E8 | | 00000780 CC CC B7 CC CC CC CC CC CC CC CC CC 47 CC 11 t11111611161 | |
| | | | 000007c0 cc cc cc cc cc cc 04 cc cc c5 5 cc cc cc cc 1111111.11111111 | |
| 00000700 | | | 000007D0 CC CC CC 2D CC CC CC CC CC CC CC CC CC 8A CC 111-11111111111111 | |
| | | | 000007E0 CC CC CC B8 CC CC CC CC CC AF CC CC CC CC CC III. IIIIII IIIII | |
| 000007F0 | | | 000007F0 cc cc cc cc cc cc cc cc p7 cc cc cc cc cc cB llllllll. | |
| | | | 00000800 cc cc cc c2 27 cc cc cc c53 cc cc cc cc cc cc illii'illisillili | |
| | | | 00000810 cc 2A cc | |
| 00000820 | | | 00000820 cc si cc cc cc ilililililililililililililili | |
| 00000830 | | | 00000830 CC CC CC CC CC E3 CC CC CC CC CC CC CC C1 CC 111111a11111111 | |
| 00000840 | | | 00000840 cc 32 cc | |
| 00000850 | | | 00000850 cc | |
| | | 1.1 | | |

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:

wOloksMk.exe 3864 McRegSetInfoKey HKCU\Control Panel\International\Geo

Figure 14: Checks registry geolocation of the machine

| Unauthorized or pirated software has been detected. Your system has been blocked. | | | | | | | |
|--|--|--|--|--|--|--|--|
| Wilful copyright infringement is a federal crime that carries penalties of up to five years in federal prison, a \$250,000 fine, forfeiture and restitution (17 U.S.C s.506, 18 U.S.C s.2319) | | | | | | | |
| As a first-time offender you are required by law to pay a fine of 150 GBP If the fine is not paid within three days, a warrant will be issued for your arrest, which will be forwarded to your local authorities. You will be charged, fined, convicted for up to 5 years. There are two ways to pay a fine: 1. You can pay your fine online through BitCoin. BitCoin is available nationwide. Click the tabs below to find the nearest ATM or exchange. Your computer will be unlocked after you make your payment. 2. (Offline Option) You can come to your local courthouse and pay your fine at the 'Cashiers' window. Your computer will be unlocked within 4-5 working days. To regain access now, transfer BitCoin to the following address (click to copy): | | | | | | | |
| 1JXum7vGYaUeWZadJrZHGE4tnAQZm8esd3 Online fine payments are securely processed by | | | | | | | |
| After the payment is finalized enter Transfer ID below. LLOYDS BANK | | | | | | | |
| Amount: Transfer ID: | | | | | | | |
| BTC 0.909 PAY FINE | | | | | | | |
| NOTE: Files on this computer, including network files, have been encrypted and disabled. Files will be restored after the fine is paid. Do not attempt to remove this message. This will damage your files, hardware and Windows installation beyond recovery. <u>View encrypted files</u> | | | | | | | |
| Payment How to pay a fine Find nearest ATM Online Exchanges Internet Browser Notepad Network Connections | | | | | | | |
| | | | | | | | |
| Operation Global 3 is a coordinated effort by U.S., U.K., Canadian and European law enforcement agencies targeting computers with pirated content. | | | | | | | |

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 <u>\$13,000 USD per incident</u>, 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.

<u>Back</u>