

DENTRO L'ALVEARE

alla ricerca del nettare perduto

Riccardo Di Pietro
malware analyst, security researcher

Who am I

- Ho 37 anni ma già dopo pochi anni di vita ho dato i miei primi "format c:" sul 386 di mio padre.
- Appassionato da sempre di tutto ciò che avesse una lucina che si accende e si spegne!
- Da pentester con una laurea in Ingegneria Elettronica mi sono innamorato del reverse engineering occupandomi quotidianamente di malware analysis. Sono il responsabile della mlw analysis in Deloitte RA.



@reecdeep (Twitter)



linkedin.com/in/riccardodp

- Introduzione
- Il gruppo Hive
- L'analisi
- Il keystream decryptor
- Il file decryptor

Introduzione

In un contesto storico che assiste ad un netto aumento degli attacchi informatici ad aziende di tutto il mondo, è inevitabile porre la giusta attenzione all'analisi degli operatori ransomware e dei loro prodotti.



Joe Stocker
@ITguySoCal

Conti's ransomware attack on the Costa Rica Gov began with valid flat network giving access to misconfigured administrative shares. Most networks vulnerable to the same sequence. The ZeroTrust+Microsegment is urgent.



bleepingcomputer.com

How Conti ransomware hacked and encrypted the Costa Rican government, showing the attack's precision and the speed of moving from...

8:08 AM · Jul 22, 2022 · Twitter for iPhone



Hozan Murad
@HozanMurad · Nov 8, 2021

#MediaMarkt / #Saturn gerade scheinbar in ganz DE von #Ransomware betroffen.

Alle Kassen still, nichts läuft, sieht nicht gut aus

Server-Probleme

Alle Daten auf den Server wurden durch Trojaner verschlüsselt. Ingolstadt ist draussen arbeiten (Problem besteht bundesweit).

BITTE AUF KEINEN FALL HIER SELBST IN VERSUCHEN DATEN WIEDERHERZUSTELLEN (durch Umbenennen), damit wird alles normal gemacht und die Daten sind dann mit einem Passwort auf jeden Fall permanent weg.

Wie gesagt, Ingolstadt ist draussen am arbeiten

13

128

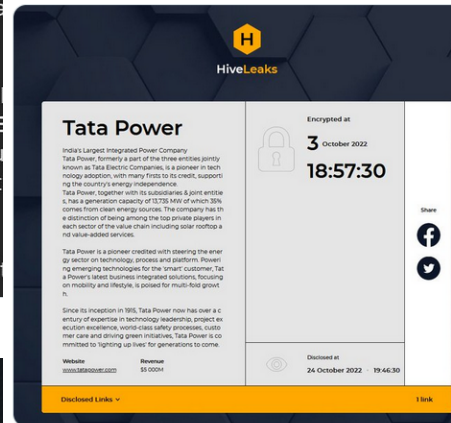
134



vx-underground
@vxunderground

HIVE ransomware group has ransomed Tata Power, a multi-billion dollar electric utility company based in Mumbai, India.

Tata's confirmed the breach. Customers and sensitive data are affected, but core functionality is present and customers still have electricity.



4:10 AM · Oct 25, 2022 · Twitter Web App

92 Retweets 6 Quote Tweets 237 Likes

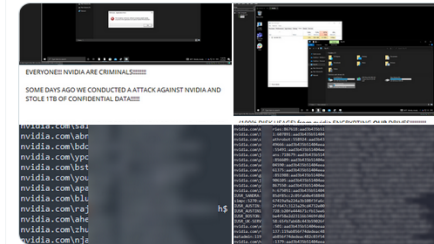


Sourfiane Tahiri
@Souf14n3

Seriously what the hell is going on !!
#Lapsus is claiming attacks on #Nvidia then #Nvidia hit back with a #Ransomware

Lapsus claims to have 1TB of data and is leaking all Nvidia employees' passwords and NTLM hashes

@Cyberknow20
@SOSIntel
@vxunderground
@ransomwaremap



7:19 AM · Feb 26, 2022 · Twitter Web App

360 Retweets 178 Quote Tweets 1,260 Likes

Il reverse engineering dei ransomware attualmente in circolazione consente di avere una panoramica sull'operatività di questi gruppi criminali.

E anche delle loro vulnerabilità.

Il lavoro presentato in queste slide è il frutto di mesi di studio sul ransomware Hive, che ha portato ad identificare diverse vulnerabilità nel codice rendendo possibile la decifrazione delle chiavi usate durante le operazioni di cifratura dei file.

In particolare il ransomware utilizza un algoritmo per la creazione delle chiavi di cifratura che, nonostante la presenza di una componente "temporale" (pensata per comportarsi come un generatore di numeri casuale), genera invece vettori di byte predicibili.

Introduzione

In passato altri malware come GandCrab, Sodinokibi, Clon, Phobos e LooCipher sono stati analizzati attentamente da alcuni ricercatori, rivelando un complesso intreccio di algoritmi.

		File Encryption	Encryption of File Encryption Key	Decryptable
Gandcrab v5.0~v5.2	Algorithm	Salsa20	RSA-2048	Exposed author's private key
	Encryption API	implemented by author	CryptEncrypt	
	Key Generation API	CryptGenRandom	CryptGenKey	
Sodinokibi	Algorithm	Salsa20	AES-256-CTR	X
	Encryption API	implemented by author		
	Key Generation API	ECDH implemented by author		
Clon	Algorithm	RC4	RSA-1024	X
	Encryption API	implemented by author	CryptEncrypt	
	Key Generation API	CryptGenKey	Used hard-coded public key	
Phobos	Algorithm	AES-256-CBC	RSA-1024	X
	Encryption API	CryptEncrypt	Used external library	
	Key Generation API	CryptGenRandom	Used hard-coded public key	
LooCipher	Algorithm	AES-128-ECB	X	Used vulnerable random generator
	Encryption API	Rijndael::Enc::AdvancedProcessBlocks in Crypto++ Library	X	
	Key Generation API	rand	X	

Introduzione

L'utilizzo della chiave asimmetrica scoraggia ogni tentativo di decifrare i file senza la chiave privata del TA. A meno che venga divulgata come nel caso di GandCrab.

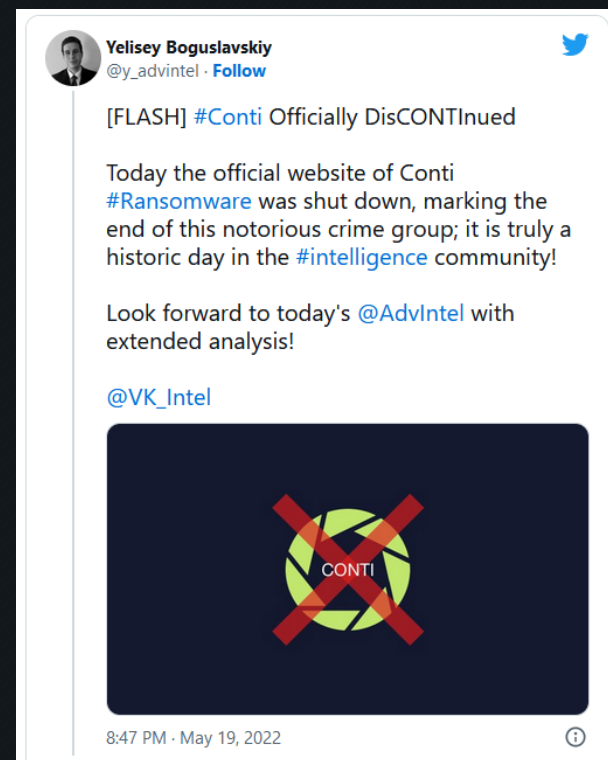
		File Encryption	Encryption of File Encryption Key	Decryptable
Gandcrab v5.0~v5.2	Algorithm	Salsa20	RSA-2048	Exposed author's private key
	Encryption API	implemented by author	CryptEncrypt	
	Key Generation API	CryptGenRandom	CryptGenKey	
Sodinokibi	Algorithm	Salsa20	AES-256-CTR	X
	Encryption API	implemented by author		
	Key Generation API	ECDH implemented by author		
Clon	Algorithm	RC4	RSA-1024	X
	Encryption API	implemented by author	CryptEncrypt	
	Key Generation API	CryptGenKey	Used hard-coded public key	
Phobos	Algorithm	AES-256-CBC	RSA-1024	X
	Encryption API	CryptEncrypt	Used external library	
	Key Generation API	CryptGenRandom	Used hard-coded public key	
LooCipher	Algorithm	AES-128-ECB	X	Used vulnerable random generator
	Encryption API	Rijndael::Enc::AdvancedProcessBlocks in Crypto++ Library	X	
	Key Generation API	rand	X	

Introduzione

Se dovesse essere utilizzato un generatore di numeri casuali debole, la chiave di cifratura potrebbe essere recuperata e i file decifrati, come nel caso di LooCipher.

		File Encryption	Encryption of File Encryption Key	Decryptable
Gandcrab v5.0~v5.2	Algorithm	Salsa20	RSA-2048	Exposed author's private key
	Encryption API	implemented by author	CryptEncrypt	
	Key Generation API	CryptGenRandom	CryptGenKey	
Sodinokibi	Algorithm	Salsa20	AES-256-CTR	X
	Encryption API	implemented by author		
	Key Generation API	ECDH implemented by author		
Clon	Algorithm	RC4	RSA-1024	X
	Encryption API	implemented by author	CryptEncrypt	
	Key Generation API	CryptGenKey	Used hard-coded public key	
Phobos	Algorithm	AES-256-CBC	RSA-1024	X
	Encryption API	CryptEncrypt	Used external library	
	Key Generation API	CryptGenRandom	Used hard-coded public key	
LooCipher	Algorithm	AES-128-ECB	X	Used vulnerable random generator
	Encryption API	Rijndael::Enc::AdvancedProcessBlocks in Crypto++ Library	X	
	Key Generation API	rand	X	

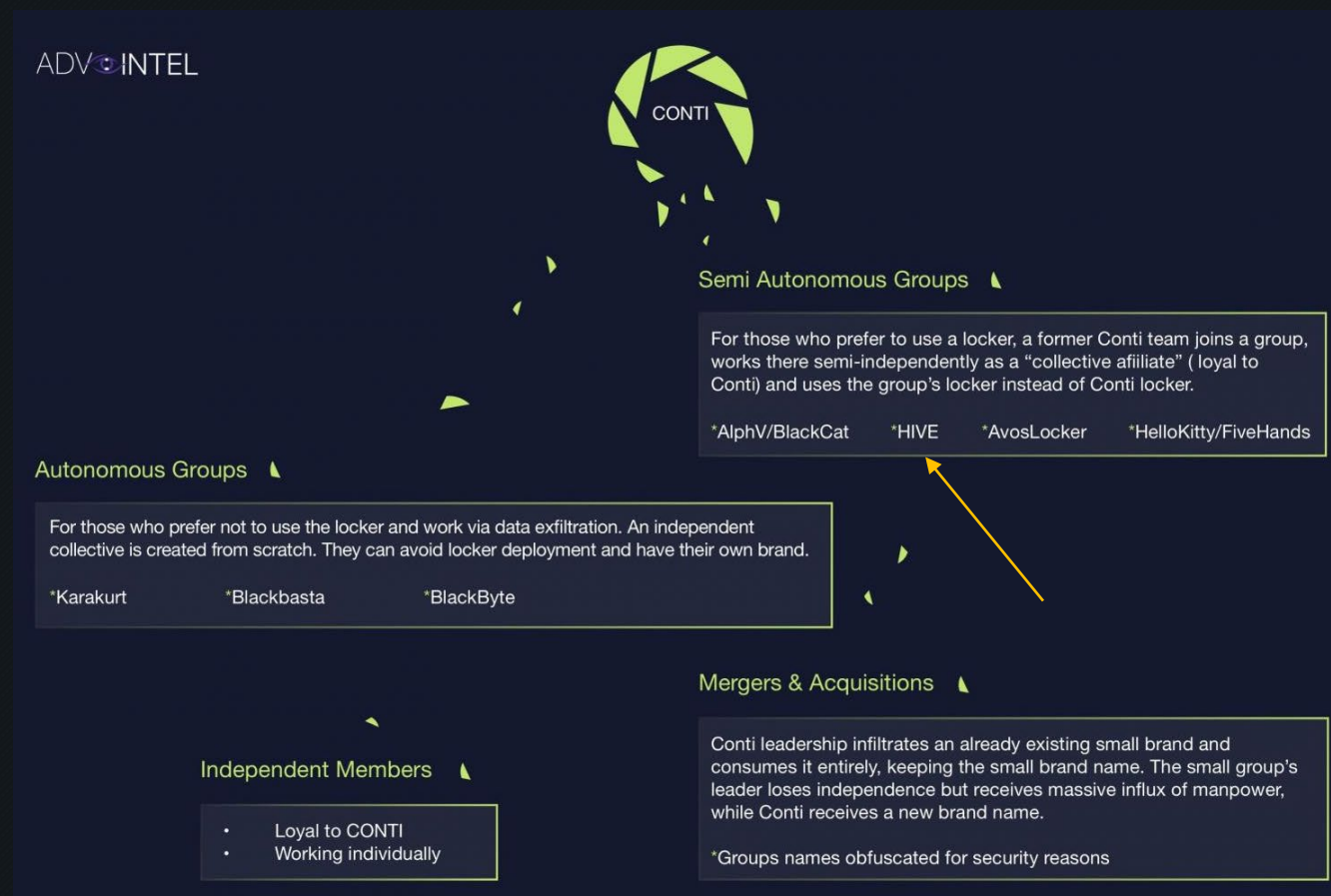
Le origini di Hive vanno ricercate nella chiusura delle operazioni di Conti a Maggio 2021



Fonte:

<https://www.bleepingcomputer.com/news/security/conti-ransomware-shuts-down-operation-rebrands-into-smaller-units/amp/>

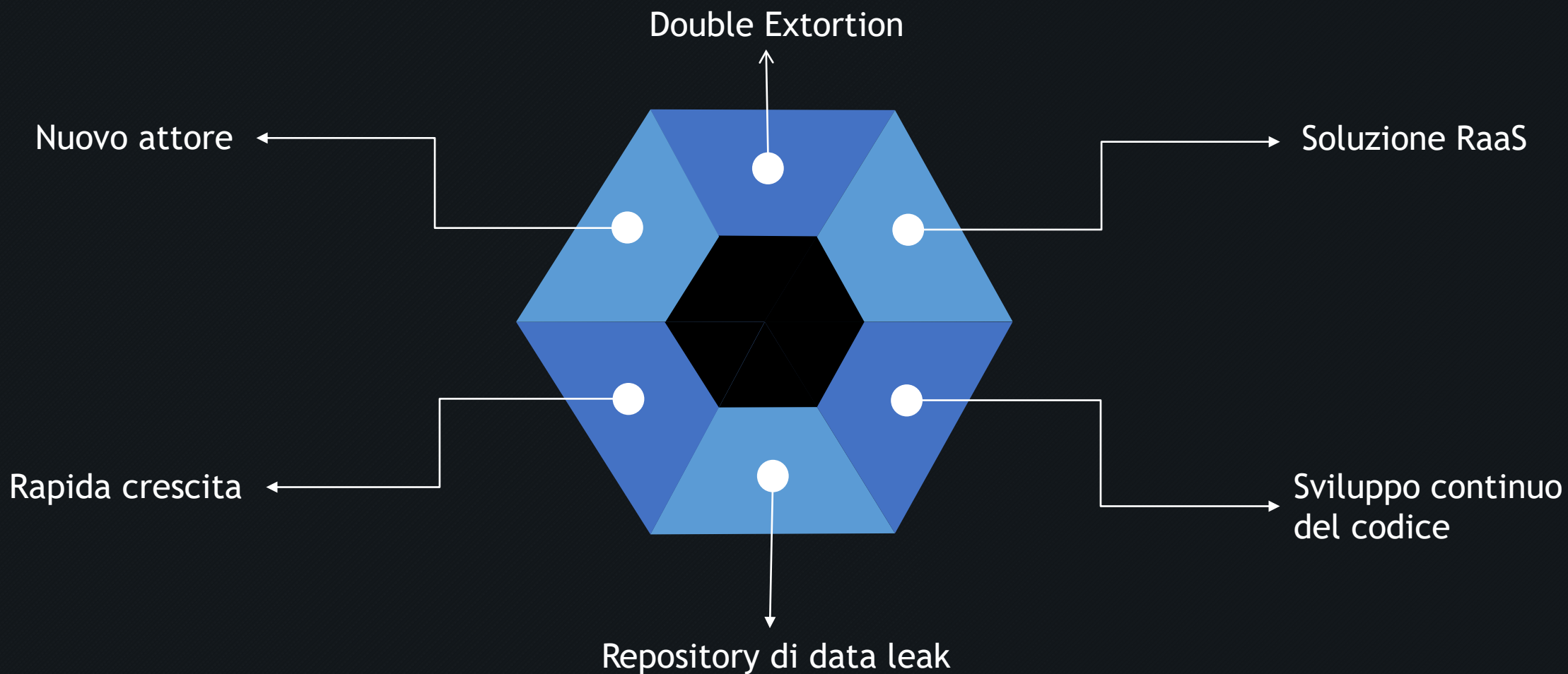
Le operazioni di smantellamento hanno dato origine ad una moltitudine di nuovi gruppi che operano come RaaS con “una nuova struttura organizzativa più orizzontale e decentrata rispetto alla precedente rigida gerarchia di Conti” (Yelisey Bogusalvskiy & Vitali Kremez)




Fonte:

<https://www.bleepingcomputer.com/news/security/conti-ransomware-shuts-down-operation-rebrands-into-smaller-units/amp/>

<https://www.advintel.io/post/discontinued-the-end-of-conti-s-brand-marks-new-chapter-for-cybercrime-landscape/>





La soluzione RaaS:



OnLine

Market → Soft

[RaaS] Affiliate program

 kkk User

 Published 09/07/2021 at 19:32

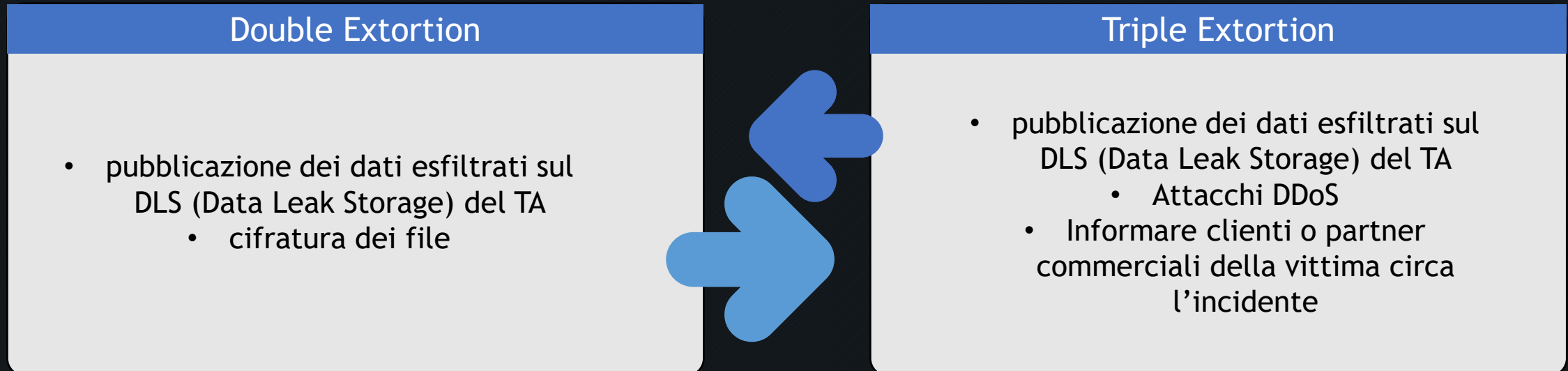
 2  115

A user with the username kkk was looking for "pentesters with their own networks". The message says that the ransomware works on Windows, Linux, FreeBSD, etc. The affiliate program has an administrative panel in TOR. Payments are made to affiliates' wallets. 80% of the payment goes to the affiliate, 20% to the creator of the affiliate program.

A RAMP user created a thread asking about what affiliate programs were currently popular and have a good reputation, to which the user with the username kkk left a comment saying that they "had a good option". This implies that they themselves had access to an affiliate program.

Fonte:
<https://blog.group-ib.com/hive>

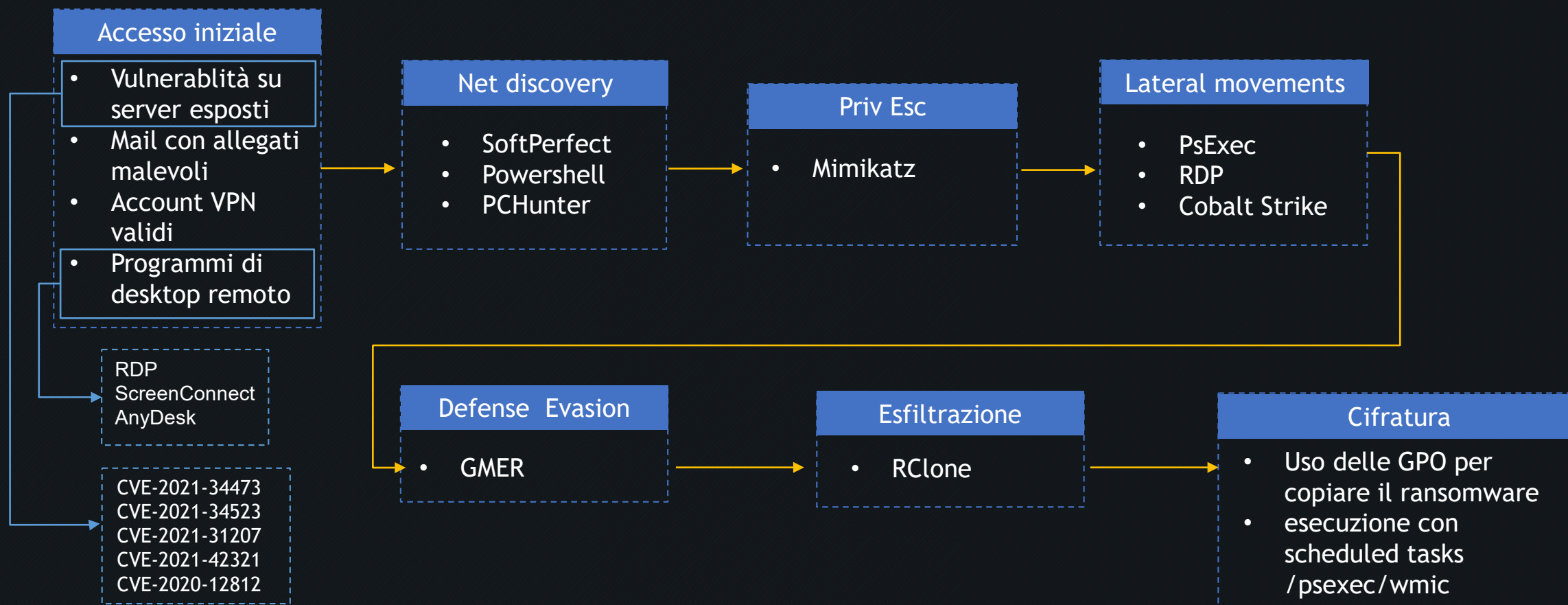
Un nuovo modello di estorsione: Triple Extortion:



Fonte:

<https://www.cisa.gov/uscert/ncas/alerts/aa22-040a>

Le TTP del gruppo Hive:



Fonte:

https://www.trendmicro.com/it_it/research/22/i/play-ransomware-s-attack-playbook-unmasks-it-as-another-hive-aff.html

<https://www.cisa.gov/uscert/ncas/alerts/aa22-321a>

Dalla richiesta di riscatto al decryptor:



Il gruppo HIVE

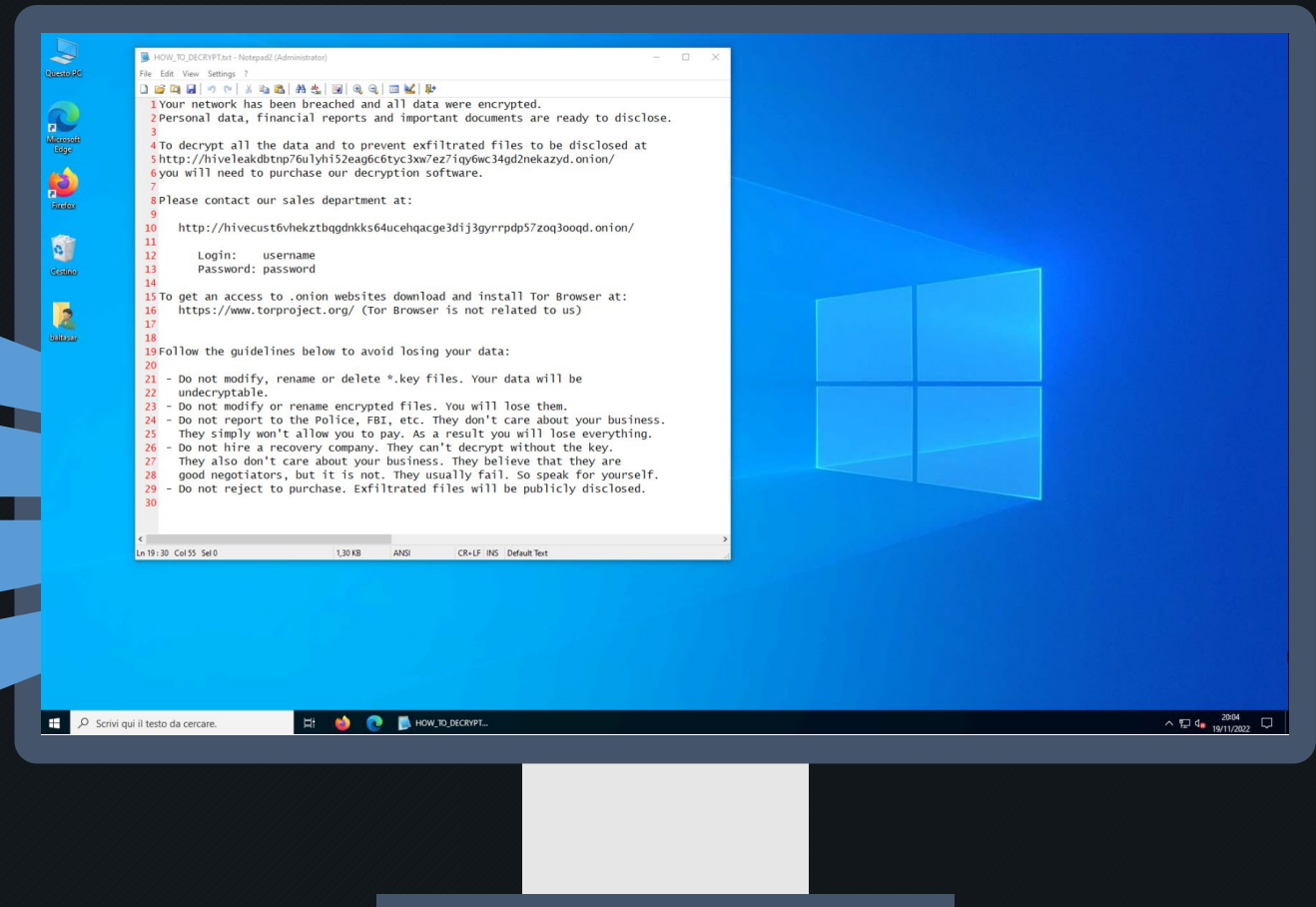
La richiesta del riscatto:

Richiesta di riscatto per la
cifratura

Richiesta di riscatto per i
dati esfiltrati

Invito ad iniziare le
trattative via chat privata

I toni dell'affiliato Hive
sono poco formali



Il gruppo HIVE

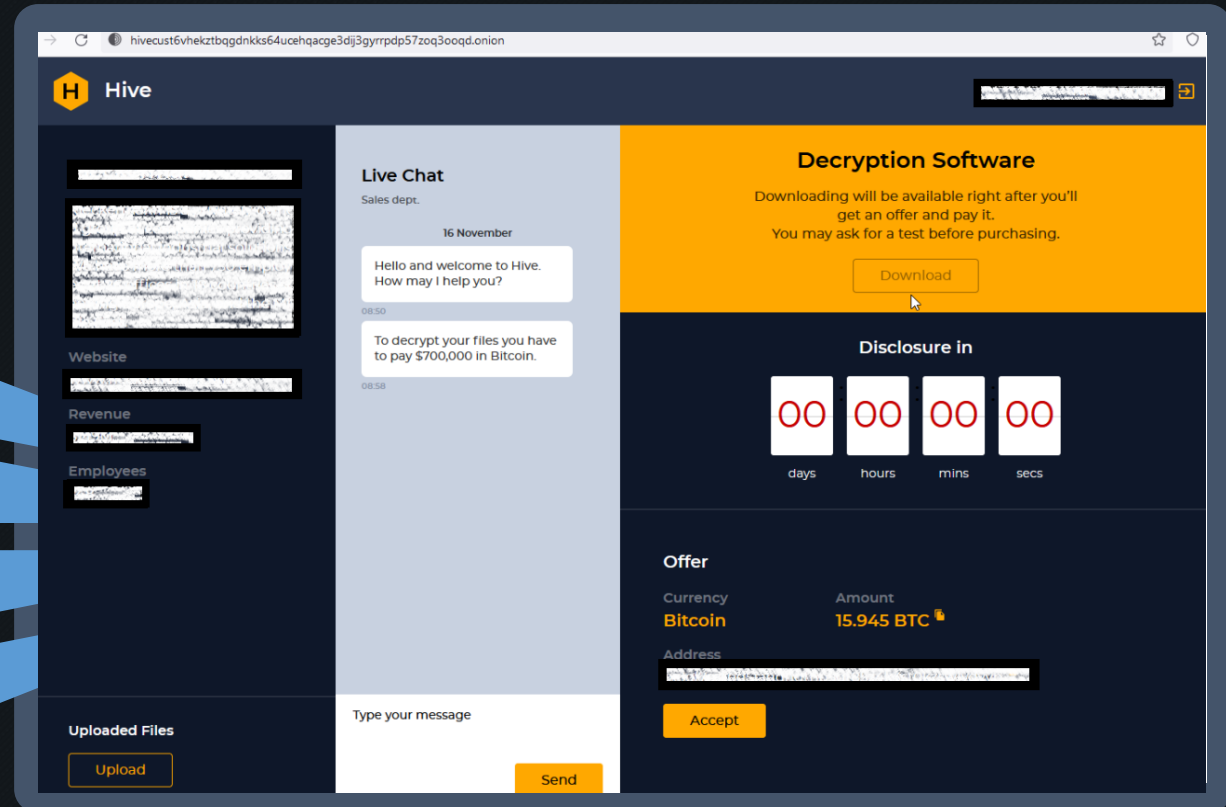
Accesso alla chat:

Pannello con breve
descrizione della vittima

Pannello per accettare
l'offerta dell'affiliato

Area per l'upload di file
come prova di decifrazione

Conto alla rovescia come
ulteriore pressione



Creazione della chat da parte dell'affiliato:

Form per l'inserimento dei
dati societari della vittima

Form per l'inserimento del
riscatto (offerta)

Creazione automatica dei
sample di ransomware

Form per l'inserimento di
eventuali link a leak

The screenshot displays the Hive web application interface. On the left, a sidebar contains navigation links: 'Overview', 'Companies', and 'Payouts'. The main content area is titled 'Companies' and features a 'Create' button. A modal window titled 'New Company' is open in the center, containing the following form fields: 'Title' (with an asterisk indicating it is required), 'Description', 'Tax Number', 'Website', 'Revenue', and 'Employees'. An orange 'Create' button is located at the bottom of the modal. The top of the interface shows a header with the Hive logo, a balance display for BTC and XMR, and a 'Pay out' button.

La trattativa in chat:

Il riscatto è l'1 -1,5% del reddito annuo dell'azienda

In alcuni casi l'affiliato di Hive ha applicato sconti fino all'80% pur di incassare

Le trattative possono essere condotte dalle vittime stesse o da negoziatori.

In un caso l'affiliato di Hive ha ceduto il 10% del pagamento del riscatto al negoziatore.

Hi are you there?? I know we are contacting to you after long time but we need your help in decrypting some of the important files. We want to decrypt some 100 files. Can you please tell us, how much will you charge to decrypt 100 files?

08:55


There is no difference between number of files. The price is for encryptor itself.

Fonte:

https://s3.amazonaws.com/talos-intelligence-site/production/document_files/files/000/095/787/original/ransomware-chats.pdf?1651576098

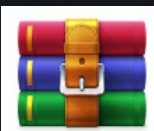
Pagamento del riscatto e download del decryptor:

Decryption Software




Congratulations!
You've made the right choice.
Run the software on every encrypted operating system in your network.
Please feel free to contact us if you need any further information.

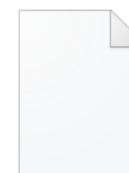
[Download](#)



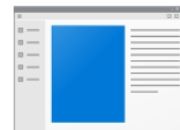
decryptor.zip



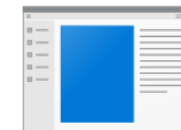
esxi_decrypt



linux_decrypt



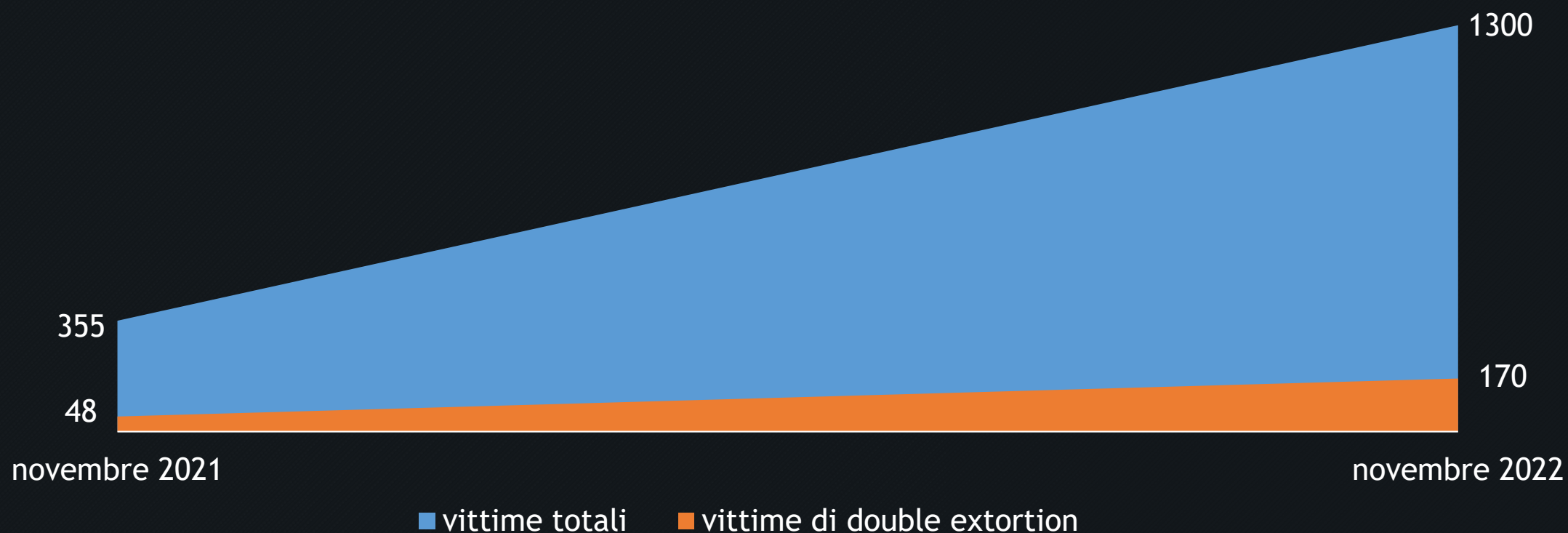
windows_x32_decrypt.exe



windows_x64_decrypt.exe

Il gruppo HIVE

Vittime del gruppo Hive nell'ultimo anno:



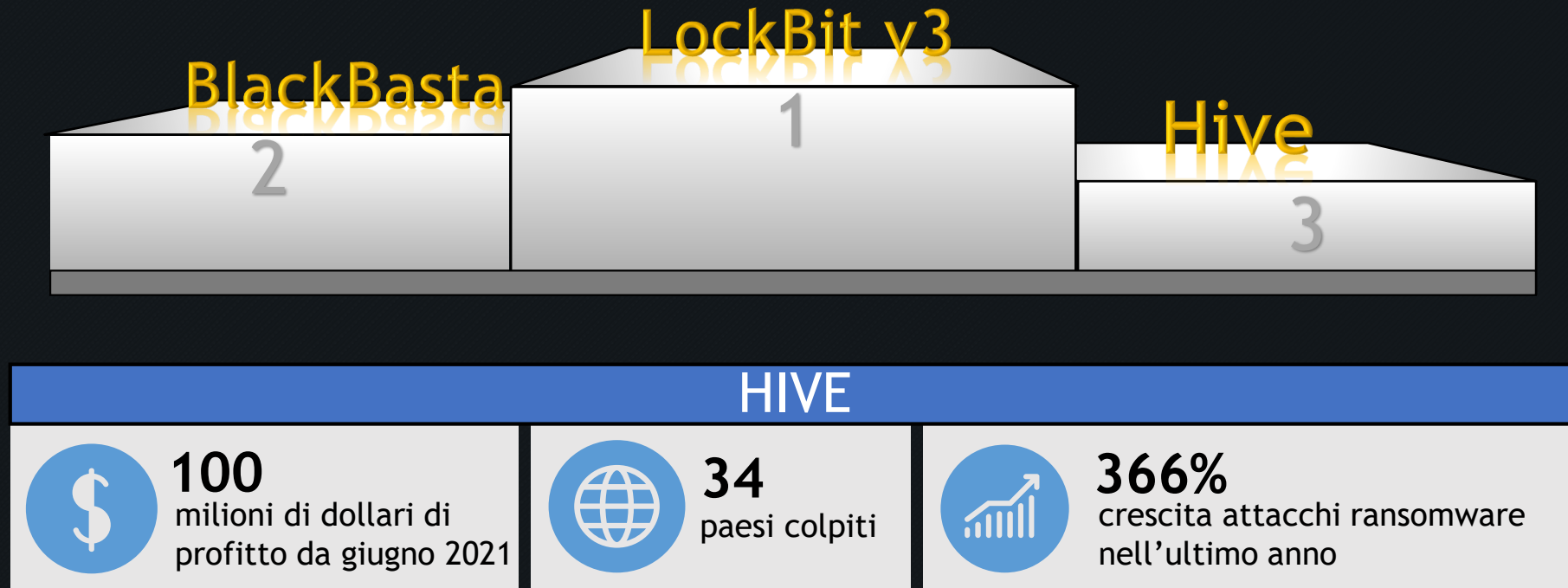
Fonti:

https://www.cisa.gov/uscert/sites/default/files/publications/aa22-321a_joint_csa_stopransomware_hive.pdf

<https://blog.group-ib.com/hive>

Il gruppo HIVE

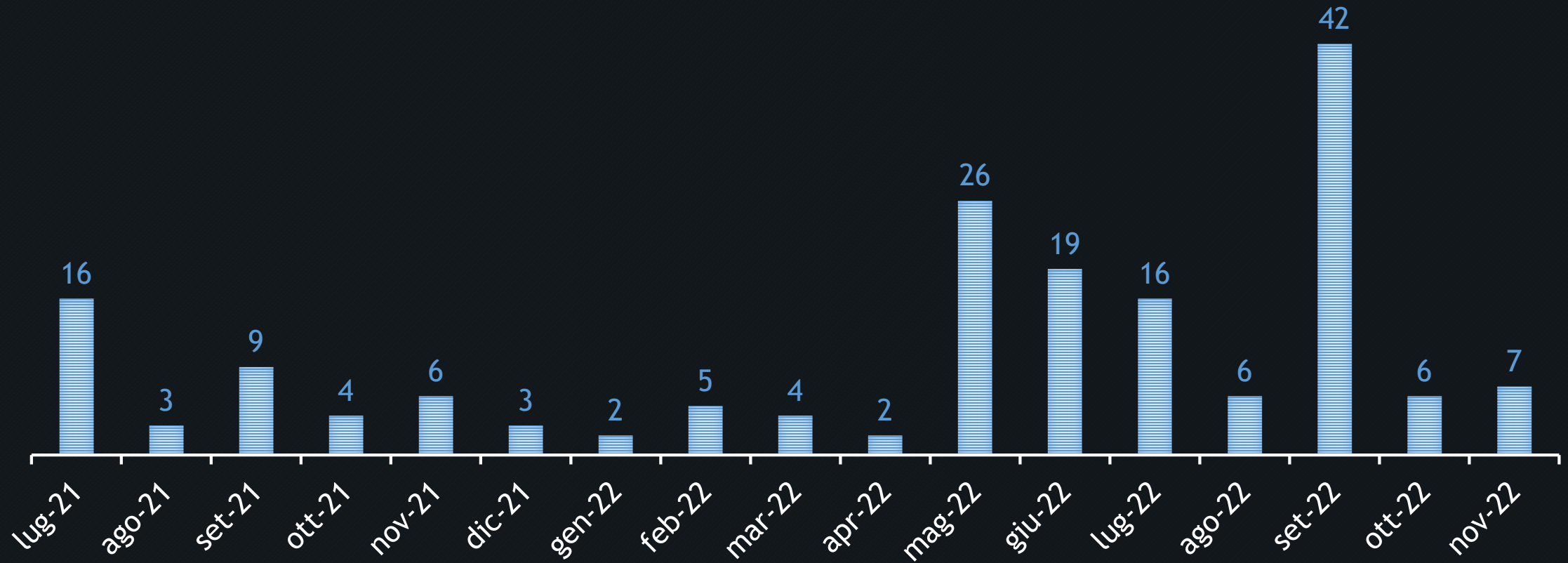
Hive è tra i gruppi RaaS più attivi del 2022:



Fonte:

<https://intel471.com/resources/whitepapers/leading-ransomware-variants-q3-2022>

Vittime del double extortion rilevate dal pannello DLS (Data Leak Storage):



Fonte:
<http://hiveleakdbtnp76ulyhi52eag6c6tyc3xw7ez7iqy6wc34gd2nekazyd.onion/>

Distribuzione geografica delle vittime del double extortion:

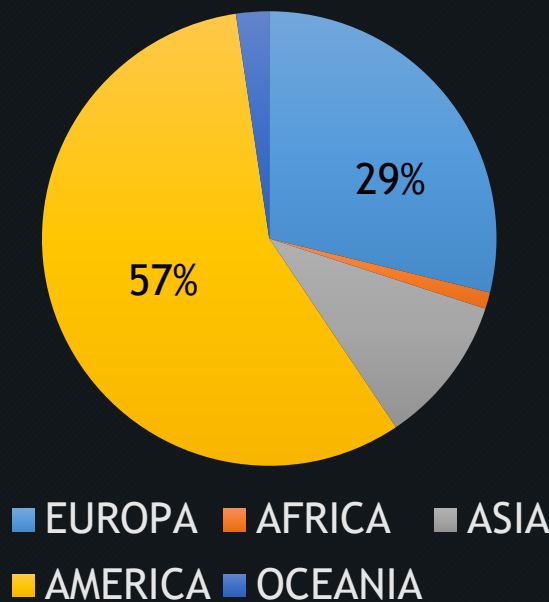
USA	Cina	Australia	India	Perù	Malesia	Norvegia	Belgio	Messico
UK	Brasile	Argentina	Paesi bassi	Africa	Mongolia	Giamaica	Irlanda	Corea del Sud
Spagna	Svizzera	Portogallo	Francia	Indonesia	Grecia	Emirati Arabi	Repubblica Dominicana	
Germania	Italia	Canada	Thailandia	Colombia	Austria	Turchia	Olanda	



Fonte:

<http://hiveleakdbtnp76ulyhi52eag6c6tyc3xw7ez7iqy6wc34gd2nekazyd.onion/>

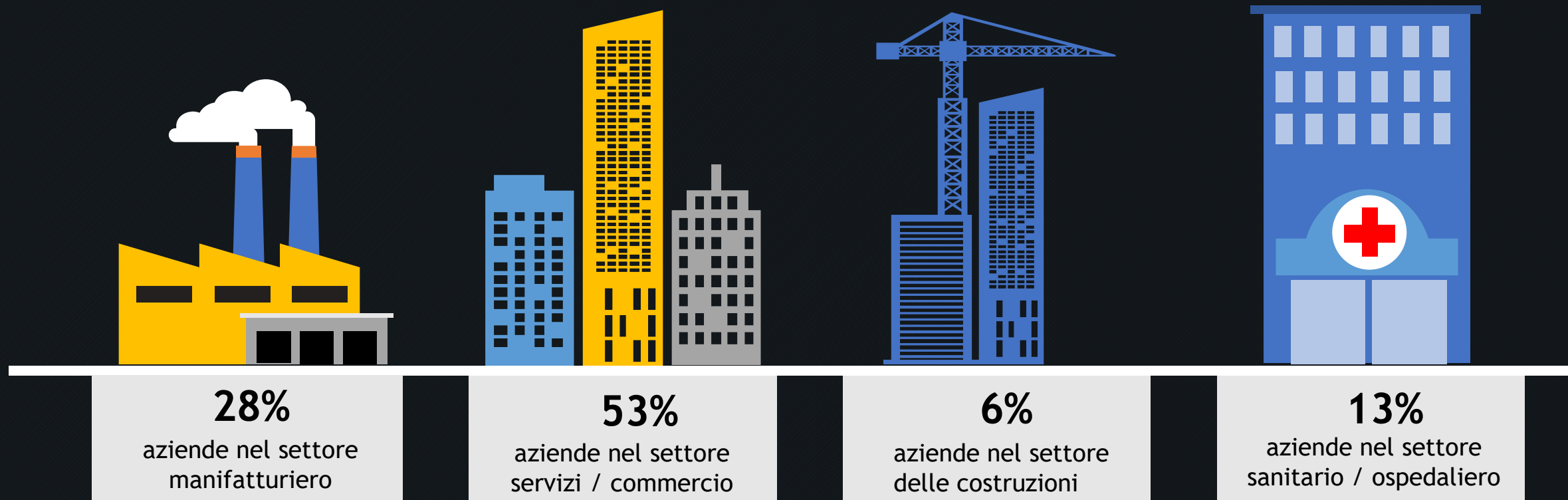
Classifica dei paesi più colpiti dal double extortion di Hive:



Fonte:

<http://hiveleakdbtnp76ulyhi52eag6c6tyc3xw7ez7iqy6wc34gd2nekazyd.onion/>

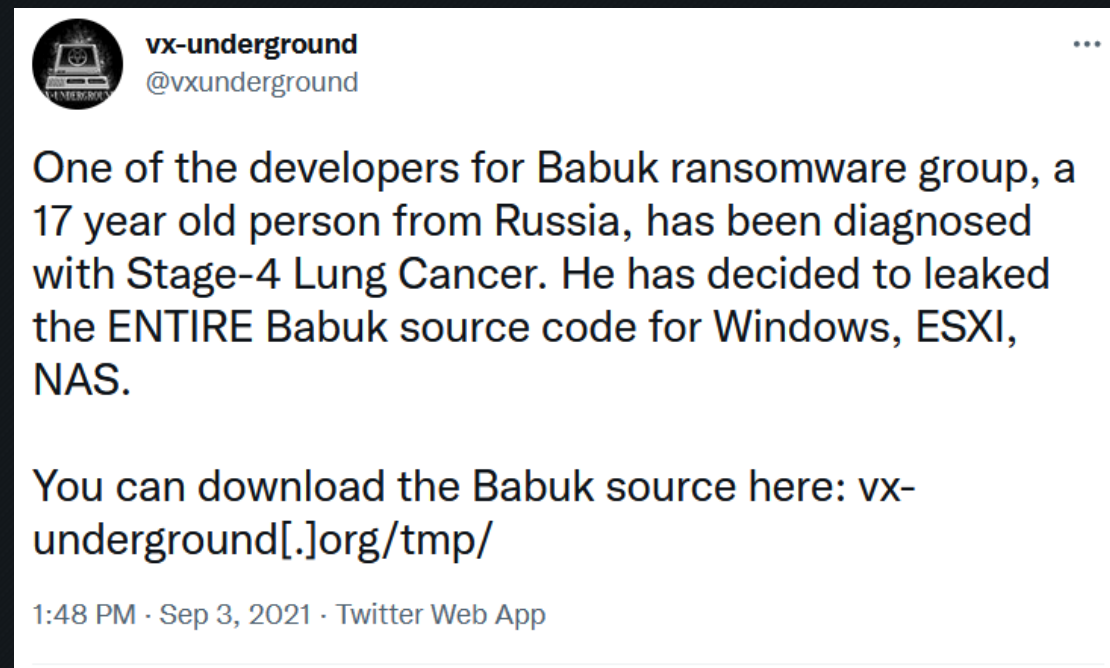
Le attività più colpite dal double extortion di Hive



Fonte:

<https://www.spiceworks.com/it-security/security-general/articles/why-hive-ransomware-is-dangerous/>

Il codice sorgente di BABUK fornisce un ulteriore punto di vista



Fonte:

<https://twitter.com/vxunderground/status/1433758742244478982>

Analogie tra il codice sorgente di BABUK ed il binario di HIVE

HIVE

Address	Disassembly	Assembly
00410F3C	85F6	test esi,esi
00410F3E	74 41	je cryptor.410F81
00410F40	8B10	mov edx,dword ptr ds:[eax]
00410F42	8D78 0C	lea edi,dword ptr ds:[eax+C]
00410F45	8D4C24 40	lea ecx,dword ptr ss:[esp+40]
00410F49	FF70 08	push dword ptr ds:[eax+8]
00410F4C	E8 43E50300	call cryptor.44F494
00410F51	83C4 04	add esp,4
00410F54	83C6 F4	add esi,FFFFFFF4
00410F57	84C0	test al,al
00410F59	89F8	mov eax,edi
00410F5B	74 DF	je cryptor.410F3C
00410F5D	FFB424 58040000	push dword ptr ss:[esp+458]
00410F64	6A 00	push 0
00410F66	6A 01	push 1
00410F68	E8 C3B80700	call <JMP.&OpenProcess>
00410F6D	85C0	test eax,eax
00410F6F	74 10	je cryptor.410F81
00410F71	89C6	mov esi,eax
00410F73	6A 00	push 0
00410F75	50	push eax
00410F76	E8 25B80700	call <JMP.&TerminateProcess>
00410F78	56	push esi
00410F7C	E8 27BA0700	call <JMP.&CloseHandle>
00410F81	8D8424 50040000	lea eax,dword ptr ss:[esp+450]
00410F88	50	push eax
00410F89	8B7424 14	mov esi,dword ptr ss:[esp+14]
00410F8D	56	push esi
00410F8E	E8 8DB80700	call <JMP.&Process32NextW>
00410F93	85C0	test eax,eax

BABUK

```
void _stop_processes() {
    HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPALL, 0);
    PROCESSENTRY32W pEntry;
    pEntry.dwSize = sizeof(pEntry);
    BOOL hRes = Process32FirstW(hSnapshot, &pEntry);
    while (hRes)
    {
        for (int i = 0; i < _countof(processes_to_stop); i++) {
            if (lstrcmpW(processes_to_stop[i], pEntry.szExeFile) == 0) {
                HANDLE hProcess = OpenProcess(PROCESS_TERMINATE, 0, (DWORD)pEntry.th32ProcessID);
                if (hProcess != NULL)
                {
                    TerminateProcess(hProcess, 9);
                    CloseHandle(hProcess);
                }
                break;
            }
        }
        hRes = Process32NextW(hSnapshot, &pEntry);
        CloseHandle(hSnapshot);
    }
}
```

Il gruppo Hive

Analogie tra il codice sorgente di BABUK ed il binario di HIVE

HIVE

```
0040F12B E8 30640400 call <JMP.&FindFirstVolumeW>
0040F130 85C0 test eax, eax
0040F132 890424 mov dword ptr ss:[esp], eax
0040F135 0F84 07010000 je cryptor.40F242
0040F138 8D4424 40 lea eax, dword ptr ss:[esp+40]
0040F13F 897C24 10 mov dword ptr ss:[esp+10], edi
0040F143 895C24 08 mov dword ptr ss:[esp+8], ebx
0040F147 B9 04010000 mov ecx, 104
0040F14C 894C24 40 mov dword ptr ss:[esp+40], ecx
0040F150 50 push eax
0040F151 51 push ecx
0040F152 53 push ebx
0040F153 56 push esi
0040F154 E8 2F630400 call <JMP.&GetVolumePathNamesForVolumeNameW>
0040F159 85C0 test eax, eax
0040F15B 74 0B je cryptor.40F168
0040F15D 837C24 40 00 cmp dword ptr ss:[esp+40], 0
0040F162 0F85 B2000000 jne cryptor.40F21A
0040F168 8B8C24 A8000000 mov eax, dword ptr ss:[esp+A8]
0040F16F 85C0 test eax, eax
0040F171 0F84 A3000000 je cryptor.40F21A
0040F177 48 dec eax
0040F178 898424 A8000000 mov dword ptr ss:[esp+A8], eax
0040F17F 8D0440 lea eax, dword ptr ds:[eax+eax*2]
0040F182 8B8C24 A0000000 mov ecx, dword ptr ss:[esp+A0]
0040F189 8B3C81 mov edi, dword ptr ds:[ecx+eax*4]
0040F18C 85FF test edi, edi
0040F18E 0F84 86000000 je cryptor.40F21A
0040F194 8B5481 04 mov edx, dword ptr ds:[ecx+eax*4+4]
0040F198 8B4481 08 mov eax, dword ptr ds:[ecx+eax*4+8]
0040F19C 8D8C24 60040000 lea ecx, dword ptr ss:[esp+460]
0040F1A3 898C24 80010000 mov dword ptr ss:[esp+180], edi
0040F1AA 895424 04 mov dword ptr ss:[esp+4], edx
0040F1AE 01F8 add eax, edi
0040F1B0 8D9424 80010000 lea edx, dword ptr ss:[esp+180]
0040F1B7 898424 84010000 mov dword ptr ss:[esp+184], eax
0040F1BE 66:C78424 88010000 00 mov word ptr ss:[esp+188], 0
0040F1C8 C78424 8C010000 010000 mov dword ptr ss:[esp+18C], 1
0040F1D3 E8 8E760000 call cryptor.416866
0040F1D8 8B9C24 60040000 mov ebx, dword ptr ss:[esp+460]
0040F1DF 56 push esi
0040F1E0 53 push ebx
0040F1E1 E8 CA610400 call <JMP.&SetVolumeMountPointW>
0040F1E6 F78424 64040000 FFFFFFFF test dword ptr ss:[esp+464], 7FFFFFFF
```

BABUK

```
drive[0] = L'\0';
if (WCHAR* volume = (WCHAR*)_halloc(32768 * sizeof(WCHAR))) {
    if (WCHAR* partition = (WCHAR*)_halloc(32768 * sizeof(WCHAR))) {
        HANDLE hFind = FindFirstVolumeW(volume, 32768);
        do {
            if (driveCounter > 0) {
                if (GetVolumePathNamesForVolumeNameW(volume, drive, driveSize, &retSize)) {
                    if (lstrlenW(drive) == 3) {
                        drive[0] = L'\0';
                        continue;
                    }
                }
                SetVolumeMountPointW(freeLetters[--driveCounter], volume);
            } else break;
        } while (FindNextVolumeW(hFind, volume, 32768));
        FindVolumeClose(hFind);
        _hfree(partition);
    }
    _hfree(volume);
}
```


Il gruppo Hive

Analogie tra il codice sorgente di BABUK ed il binario di HIVE

HIVE

Address	Disassembly	Comment
5-100B1	50	push ecx
5-100D0	E8 7C0C0400	call <cryptor.call_to_SetFilePointerEx>
5-100C8	83C4 10	add esp,10
5-100C8	837C24 20 01	cmp dword ptr ss:[esp+20],1
5-100C3	0F84 4F040000	je cryptor.404A61
5-100BD	8B5424 74	mov edx,dword ptr ss:[esp+74]
5-100B9	89D9	mov ecx,ebx
5-100B7	FF7424 10	push dword ptr ss:[esp+10]
5-100B3	FF7424 04	push dword ptr ss:[esp+4]
5-100AF	E8 CB0B0400	call <cryptor.call_to_ReadFile>
5-100AA	83C4 08	add esp,8
5-100A7	837C24 20 01	cmp dword ptr ss:[esp+20],1
5-100A2	0F84 2E040000	je cryptor.404A61
5-1009C	8B7424 24	mov esi,dword ptr ss:[esp+24]
5-10098	85F6	test esi,esi
5-10096	0F84 6D040000	je cryptor.404AAC
5-10090	31DB	xor ebx,ebx
5-1008E	89B424 B8000000	mov dword ptr ss:[esp+88],esi
5-10087	395C24 10	cmp dword ptr ss:[esp+10],ebx
5-10083	74 65	je cryptor.4046B3
5-10081	8B7424 08	mov esi,dword ptr ss:[esp+8]
5-1007D	8B7C24 14	mov edi,dword ptr ss:[esp+14]
5-10079	8D43 01	lea eax,dword ptr ds:[ebx+1]
5-10076	894424 0C	mov dword ptr ss:[esp+C],eax
5-10072	B8 00000000	mov eax,0
5-1006D	01DE	add esi,ebx
5-10068	11C7	adc edi,eax
5-10069	50	push eax
5-10068	68 00FF2F00	push 2FFF00
5-10063	57	push edi
5-10062	56	push esi
5-10061	E8 A10B0500	call cryptor.455214
5-1005C	83C4 10	add esp,10
5-10059	894424 1C	mov dword ptr ss:[esp+1C],eax
5-10055	31C0	xor eax,eax
5-10053	50	push eax
5-10052	68 00FD2F00	push 2FFD00
5-1004D	57	push edi
5-1004C	56	push esi
5-1004B	8B8424 C8000000	mov esi,dword ptr ss:[esp+C8]
5-10044	E8 840B0500	call cryptor.455214
5-1003F	83C4 10	add esp,10
5-1003C	8B4C24 18	mov ecx,dword ptr ss:[esp+18]
5-10038	8B7C24 04	mov edi,dword ptr ss:[esp+4]
5-10034	8B5424 1C	mov edx,dword ptr ss:[esp+1C]
5-10030	8A0401	mov al,byte ptr ds:[ecx+eax]
5-1002D	320417	xor al,byte ptr ds:[edi+edx]
5-1002A	8B1424	mov edx,dword ptr ss:[esp]
5-10027	30041A	xor byte ptr ds:[edx+ebx],al
5-10024	8B5C24 0C	mov ebx,dword ptr ss:[esp+C]
5-10020	39DE	cmp esi,ebx
5-1001E	75 95	jne cryptor.404648
5-1001C	8B5424 74	mov edx,dword ptr ss:[esp+74]
5-10018	8D4C24 20	lea ecx,dword ptr ss:[esp+20]
5-10014	FF7424 14	push dword ptr ss:[esp+14]

BABUK

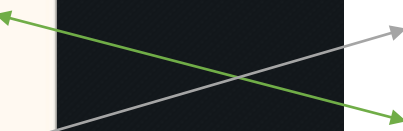
```
for (LONGLONG i = 0; i < fileChunks.QuadPart; i++) {  
    ReadFile(hFile, ioBuffer, CONST_BLOCK_PLUS, &dwRead, 0);  
    ECRYPT_process_bytes(0, &ctx, ioBuffer, ioBuffer, dwRead);  
    SetFilePointerEx(hFile, fileOffset, 0, FILE_BEGIN);  
    WriteFile(hFile, ioBuffer, CONST_BLOCK_PLUS, &dwWrite, 0);  
  
    fileOffset.QuadPart += 0xA00000i64;  
    SetFilePointerEx(hFile, fileOffset, 0, FILE_BEGIN);  
}
```

Il gruppo Hive

Analogie tra il codice sorgente di BABUK ed il binario di HIVE

HIVE		
885424 1C	mov	edx,dword ptr ss:[esp+1C]
8A0401	mov	al,byte ptr ds:[ecx+eax]
320417	xor	al,byte ptr ds:[edi+edx]
8B1424	mov	edx,dword ptr ss:[esp]
30041A	xor	byte ptr ds:[edx+ebx],al
8B5C24 0C	mov	ebx,dword ptr ss:[esp+C]
390E	cmp	esi,ebx
75 95	jne	cryptor.404648
8B5424 74	mov	edx,dword ptr ss:[esp+74]
8D4C24 20	lea	ecx,dword ptr ss:[esp+20]
FF7424 14	push	dword ptr ss:[esp+14]
FF7424 0C	push	dword ptr ss:[esp+C]
31C0	xor	eax,ecx
50	push	eax
50	push	eax
E8 B40B0400	call	<cryptor.call_to_SetFilePointerEx>
83C4 10	add	esp,10
837C24 20 01	cmp	dword ptr ss:[esp+20],1
0F84 03040000	je	cryptor.404ADD
397424 10	cmp	dword ptr ss:[esp+10],esi
8D9C24 F0000000	lea	ebx,dword ptr ss:[esp+F0]
0F82 EC070000	jbe	cryptor.404ED7
8B1424	mov	edx,dword ptr ss:[esp]
8B4C24 74	mov	ecx,dword ptr ss:[esp+74]
56	push	esi
E8 D7FF0000	call	<cryptor.call_to_WriteFile>
83C4 04	add	esp,4
8B7C24 08	mov	edi,dword ptr ss:[esp+8]
01F7	add	edi,esi
8B7424 14	mov	esi,dword ptr ss:[esp+14]
83D6 00	adc	esi,0
3C 04	cmp	al,4
0F84 D7FEFFFF	je	cryptor.4045E7
8D8424 88000000	lea	esi,dword ptr ss:[esp+88]
E9 D7030000	jmp	cryptor.404AF3
89CB	mov	ebx,ecx

BABUK
<pre>for (LONGLONG i = 0; i < fileChunks.QuadPart; i++) { ReadFile(hFile, ioBuffer, CONST_BLOCK_PLUS, &dwRead, 0); ECRYPT_process_bytes(0, &ctx, ioBuffer, ioBuffer, dwRead); SetFilePointerEx(hFile, fileOffset, 0, FILE_BEGIN); WriteFile(hFile, ioBuffer, CONST_BLOCK_PLUS, &dwWrite, 0); fileOffset.QuadPart += 0xA00000i64; SetFilePointerEx(hFile, fileOffset, 0, FILE_BEGIN); }</pre>



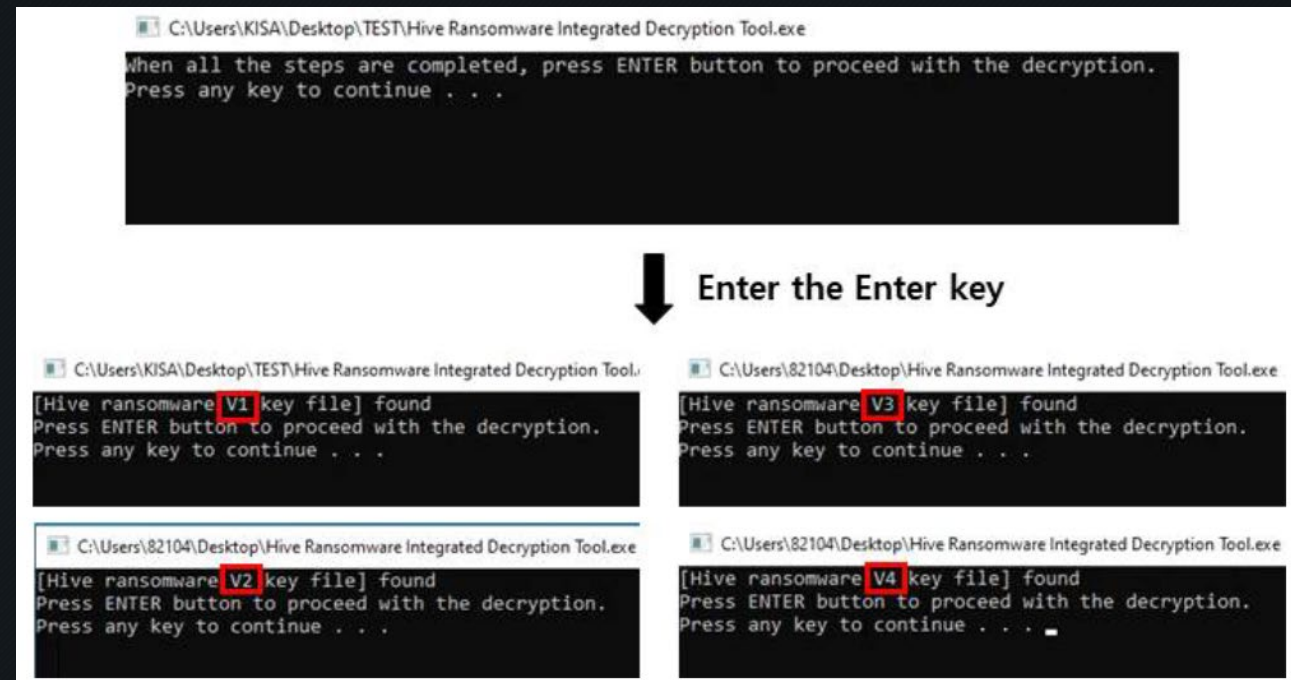
Il gruppo Hive

Le versioni del ransomware Hive



Fonte:
<https://blog.group-ib.com/hive/>

Le versioni di Hive dalla 1 alla 4 sono state analizzate da KISA ed è stato creato un tool in grado di decifrare i file key creati durante la cifratura dei file.



Fonte:

<https://arxiv.org/abs/2202.08477>

<https://seed.kisa.or.kr/kisa/Board/133/detailView.do>

L'analisi

Con Hive v5 gli sviluppatori stravolgono completamente il codice

Live Chat
Sales dept.

- The ransom payment will increase
- Your data will be placed on auction to be sold to the highest bidder
- the rest of your data will be disclosed for the public
- All client related data will be sent directly to your clients

07:45

You have a week (03/14) to make an agreement with us until the price will go up to \$10,000,000.

15:20

9 March

Btw it's an updated encryptor. There is no way to decrypt it other than to pay.

21:27

Nuovo linguaggio di programmazione

Nuovo algoritmo di cifratura

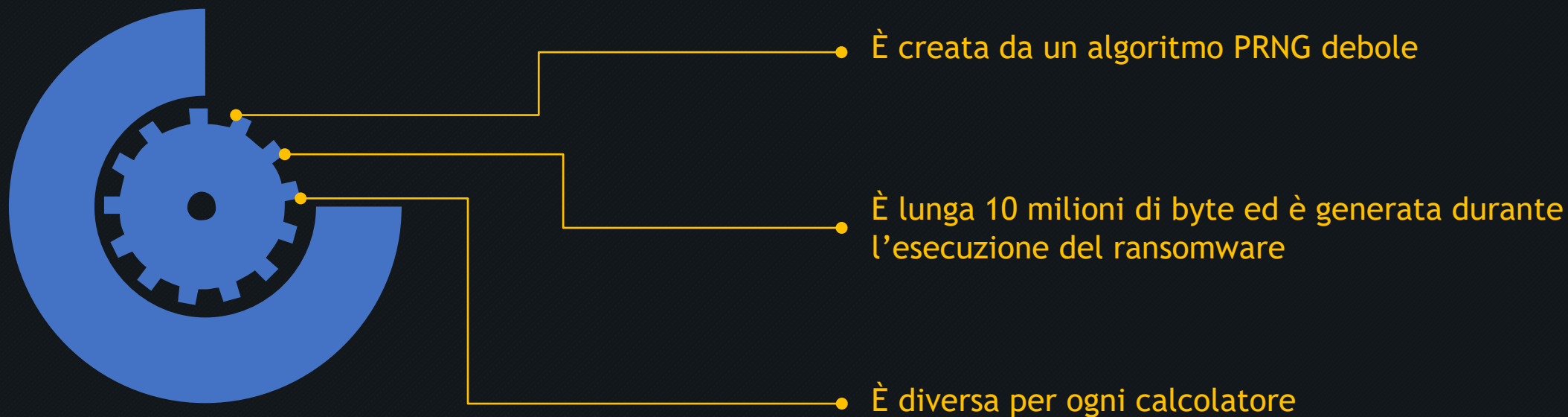
Nuova lista di processi e servizi da chiudere

Nuova chiave di cifratura dei file

Le operazioni fondamentali di Hive v5:



1. CREAZIONE DI UNA CHIAVE DI CIFRATURA

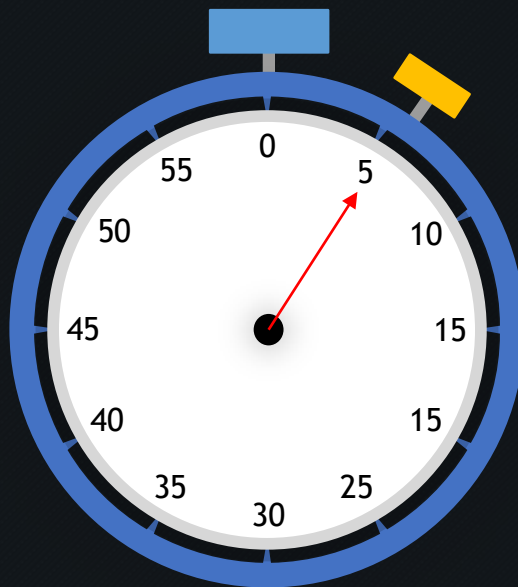


Per spiegare l'algoritmo PRNG (Pseudo Random Number Generator) immaginiamo di avere un cronometro:



L'analisi

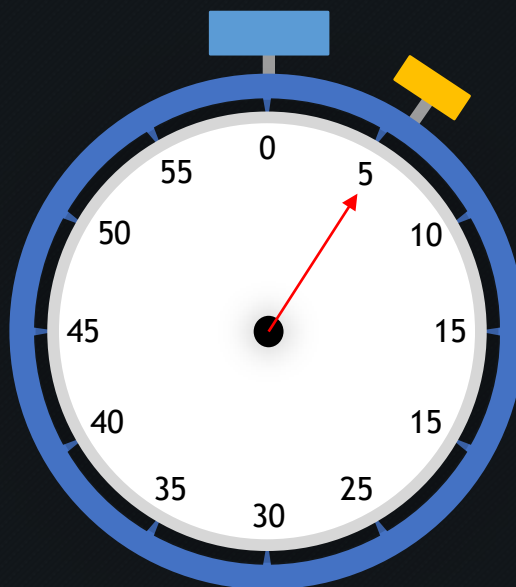
Immaginiamo di scrivere continuamente in una lista i valori indicati dalla lancetta in movimento.



L'analisi

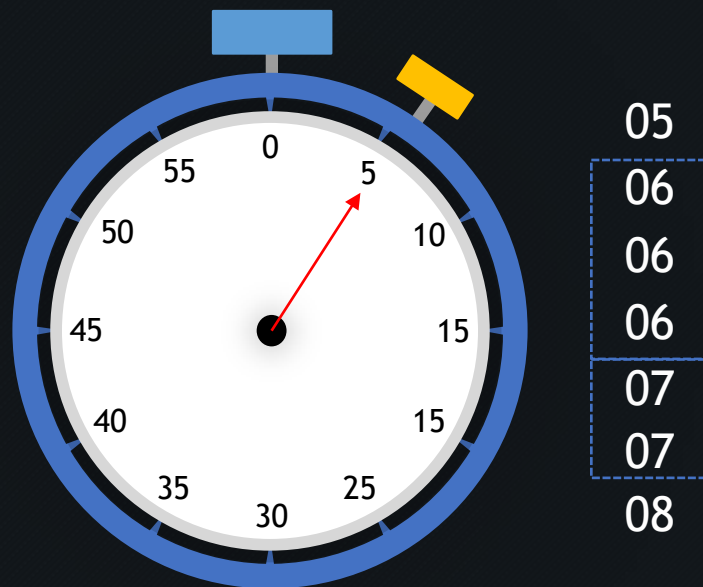
La lancetta si muove una volta al secondo.

Start!



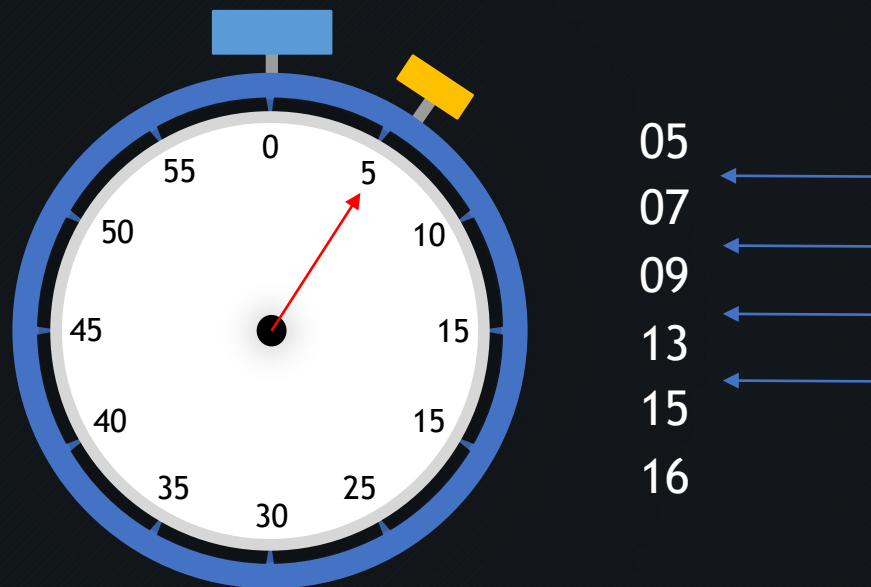
L'analisi

Se fossimo molto veloci a scrivere il valore dei secondi, potrebbe capitare di scrivere più volte lo stesso valore.



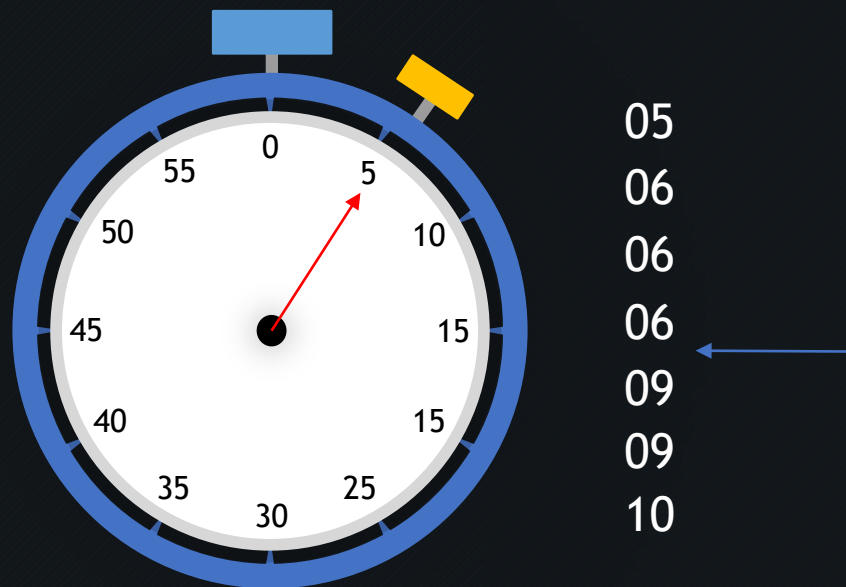
L'analisi

Al contrario se fossimo lenti, potrebbe capitare di non fare in tempo a scrivere il valore.



L'analisi

Potremmo essere anche molto veloci, MA perdere alcuni valori per distrazione



L'analisi

Se scrivessimo una volta al secondo avremmo una lista più regolare.



L'analisi

La nostra lista ha un massimo di 10 milioni di valori, raccolti con una velocità variabile. Questi si ripetono inevitabilmente. Prendendone una piccola parte abbiamo:

05	06	07	08	09	10	11	12	15	19	20	21	22	23	24	25	26	27	28	31	32	35	36
37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
00	01	02	03	04	05	06	07	08	09	10	11	12	12	15	17	19	20	21	22	23	24	25
26	27	28	29	30	31	33	34	35	36	37	38	39	40	41	42	43	44	45	46	46	46	46

Valore e frequenza del cronometro: QueryPerformanceCounter e QueryPerformanceFrequency

QPC QueryPerformanceCounter

“ È usato per leggere un istante di tempo dal cronometro ”



QPF QueryPerformanceFrequency

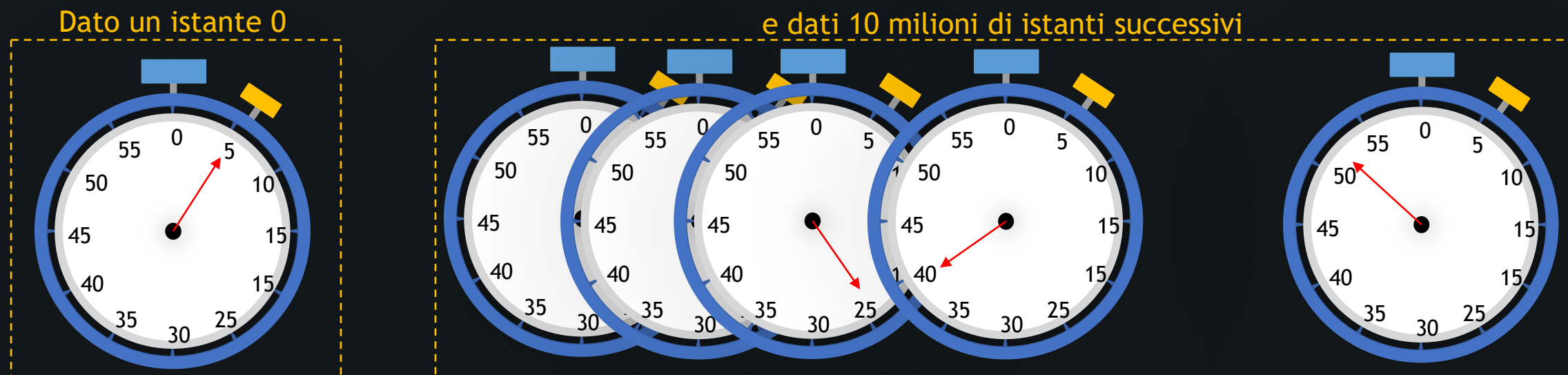
“ È la frequenza con la quale viene aggiornato il valore del cronometro ”

Fonti:

<https://learn.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancefrequency>

<https://learn.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter>

L'esempio del cronometro serve ad introdurre il reale algoritmo di creazione della chiave di cifratura:



ogni byte della chiave di cifratura è calcolato secondo la seguente relazione

$$byte_i = |istante_0 - istante_i|$$

dove i è ogni istante successivo all'istante 0

- Semplice?
- No! Non è semplice perché il cronometro usato nell'algoritmo in realtà non misura i secondi (come nell'esempio precedente) bensì i nanosecondi, quindi 9 ordini di grandezza diversi!
- Come nell'esempio del cronometro, la CPU potrebbe essere lenta, veloce, oppure veloce ma «distratta» da eventuali processi concorrenti o da un algoritmo di bilanciamento delle prestazioni (e quindi della frequenza operativa). Questo influenza la generazione della chiave di cifratura.

L'analisi

Un esempio reale di creazione dei byte:

CC	30	94	F8	5C	C0	24	88	50	B4	18	7C	E0	44	A8	0C	70	D4	38	00	64	C8	2C
90	F4	58	BC	20	84	E8	4C	B0	78	DC	40	A4	08	6C	D0	34	98	FC	60	28	8C	F0
54	B8	1C	80	E4	48	AC	10	74	D8	A0	04	68	CC	30	94	F8	5C	C0	24	88	EC	B4
18	7C	E0	44	A8	0C	70	D4	38	9C	00	C8	2C	90	F4	58	BC	20	84	E8	4C	B0	14

L'analisi

Anomalie:



CC	30	94	F8	5C	C0	24	88	50	B4	18	7C	E0	44	A8	0C	70	D4	38	00	64	C8	2C
90	F4	58	BC	20	84	E8	4C	B0	78	DC	40	A4	08	6C	D0	34	98	FC	60	28	8C	F0
54	B8	1C	80	E4	48	AC	10	74	D8	A0	04	68	CC	30	94	F8	5C	C0	24	88	EC	B4
18	7C	E0	44	A8	0C	70	D4	38	9C	00	C8	2C	90	F4	58	BC	20	84	E8	4C	B0	14

L'importanza della frequenza della CPU e dei processi simultanei!

L'analisi

Il codice semplificato per la generazione dei byte: createSeed()

```
//generating a seed
unsigned int* seed = createSeed();
//seed[0] returns EDX value - HIGHPART
//seed[1] returns EAX value - LOWPART

//dictionary_dimension 0xA00000
unsigned char byte;
unsigned int result;
for (int i = 0; i < dictionary_dimension; i++)
{
    result = createbyte(seed);
    byte = (result >> 8 * 0);
    dictionary[i] = byte;
}
```

```
unsigned int* createSeed()
{
    LARGE_INTEGER qpf;
    QueryPerformanceFrequency(&qpf);

    LARGE_INTEGER qpc;
    QueryPerformanceCounter(&qpc);

    unsigned int* out = new unsigned int[2]();
    out[0] = qpc.QuadPart / qpf.QuadPart;

    unsigned long long mod = qpc.QuadPart % qpf.QuadPart;
    out[1] = 0x3B9ACA00 * mod / qpf.QuadPart;
    //out[0] returns EDX value - HIGHPART
    //out[1] returns EAX value - LOWPART

    return out;
}
```

L'analisi

Il codice semplificato per la generazione dei byte: createByte()

```
//generating a seed
unsigned int* seed = createSeed();
//seed[0] returns EDX value - HIGHPART
//seed[1] returns EAX value - LOWPART

//dictionary_dimension 0xA00000
unsigned char byte;
unsigned int result;
for (int i = 0; i < dictionary_dimension; i++)
{
    result = createbyte(seed);
    byte = (result >> 8 * 0);
    dictionary[i] = byte;
}
```

```
//get the remainder
unsigned long long mod = (unsigned long long) qpc.QuadPart % qpf.QuadPart;
//scale factor
unsigned long long out = (unsigned long long) 0x3B9ACA00 * mod;
unsigned long long out2 = (unsigned long long) out / qpf.QuadPart;

if (out2 < seed[1])
{
    unsigned long long out3 = (unsigned long long) out2 + 0x3B9ACA00;
}
//get the "distance" from current time and the first one (seed)
unsigned long long out3 = (unsigned long long) out2 - seed[1];

return out3;
```

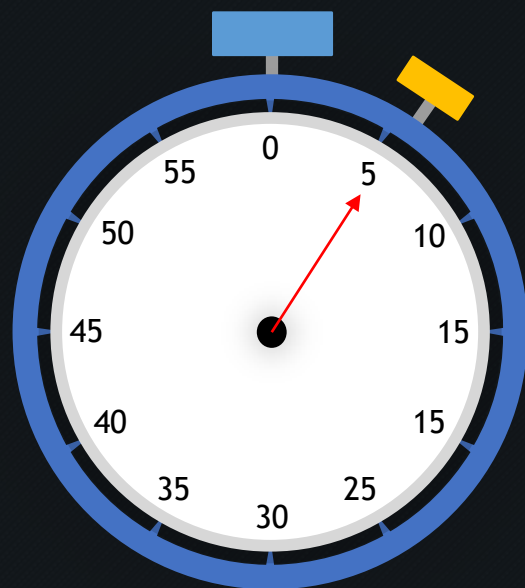

- In realtà gli sviluppatori del ransomware erano a conoscenza della scarsa randomicità del loro algoritmo
- Hanno usato diversi artifici per ritardare la lettura degli istanti di tempo e quindi tentare di aumentare l'entropia della chiave di cifratura, oltre che per rendere ulteriormente difficoltoso il reverse engineering:

- L'inserimento di JMP condizionali
- L'abuso di metodi scritti anche per fare semplici operazioni come divisione e modulo
- L'uso di fattori di scala che richiedono più di 32 bit per essere rappresentati (e quindi maggior codice da eseguire sui sistemi a 32 bit) con conseguenti sperati ritardi

```
local_30 = param_1;
uVar17 = FUN_00455174(1000000000,0,uVar5,uVar16);
uVar5 = param_2[1];
local_20 = (uint)((ulonglong)((uint)uVar17 >> 9) * 0x44b83 >> 0x20);
local_2c = *param_3;
uVar16 = param_3[1];
uVar1 = *param_2;
cVar14 = (local_2c ^ uVar1 | uVar16 ^ uVar5) != 0;
uVar7 = param_2[2];
if (uVar16 < uVar5 || uVar16 - uVar5 < (uint)(local_2c < uVar1)) {
    cVar14 = -1;
}
uVar2 = param_3[2];
cVar15 = uVar2 != uVar7;
if (uVar2 < uVar7) {
    cVar15 = -1;
}
if (cVar14 != '\0') {
    cVar15 = cVar14;
}
if (cVar15 == '\x01') {
    local_34 = local_2c - uVar1;
    uVar9 = (uVar16 - uVar5) - (uint)(local_2c < uVar1);
    if (uVar16 < uVar5 || uVar16 - uVar5 < (uint)(local_2c < uVar1)) {
```

L'analisi

L'algoritmo proposto ha il compito di creare:



1

CHIAVE DI CIFRATURA
10.485.760 byte

2

CHIAVE PRIVATA
32 byte

3

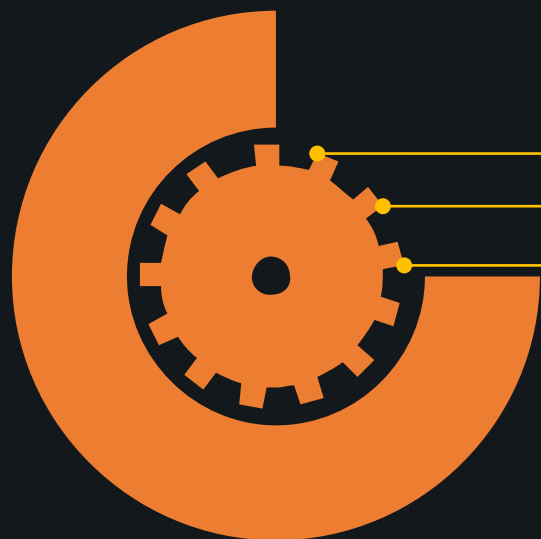
NONCE
24 byte

A cosa servono CHIAVE PRIVATA e NONCE?

Ricordiamo le operazioni fondamentali di Hive v5:



2: CIFRATURA DELLA CHIAVE



- Uso di CURVE25519 che è un algoritmo di scambio di chiavi robusto

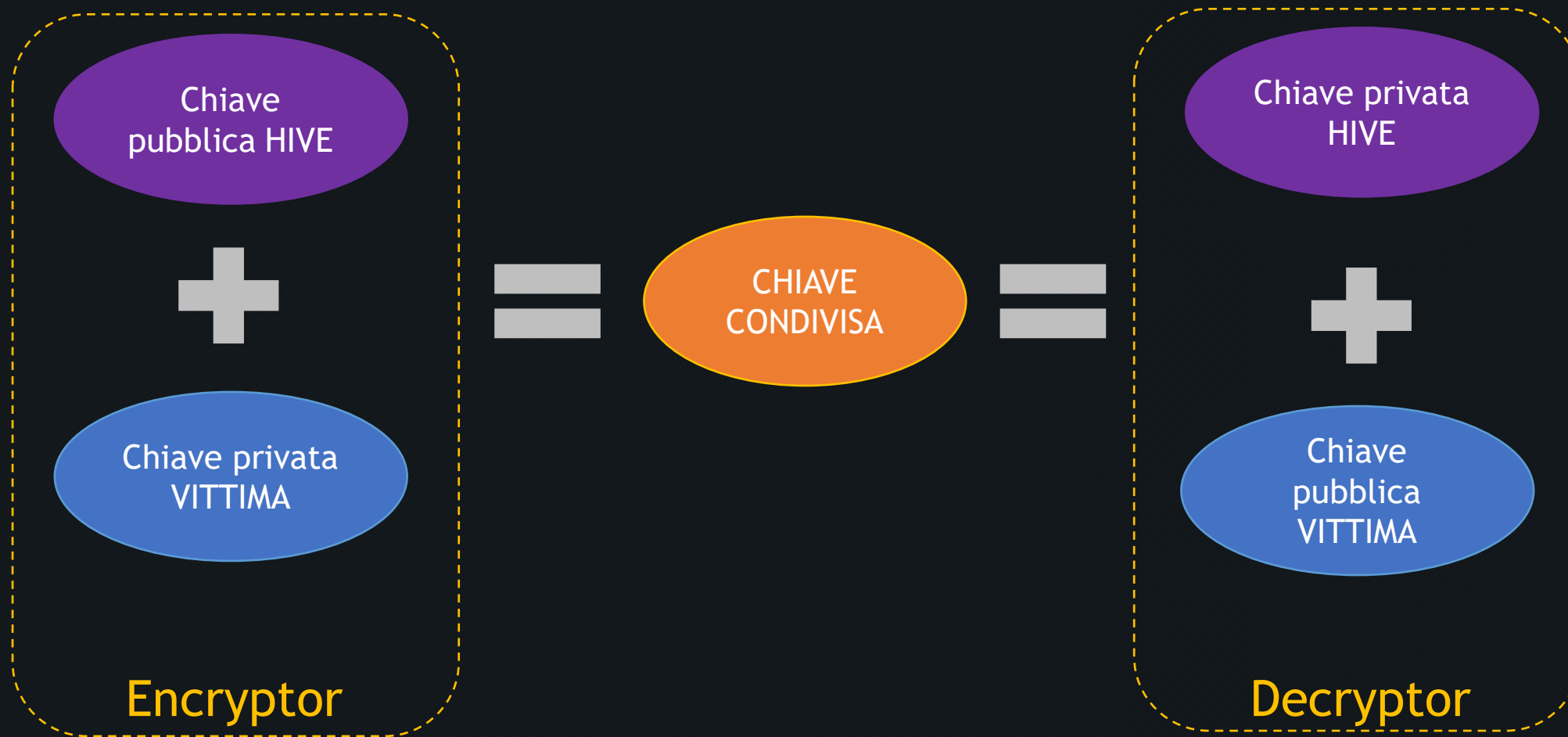
- Uso di XCHACHA20-POLY1305 che è un algoritmo di cifratura autenticato che è resistente a qualsiasi attacco attualmente conosciuto

- Uso di elementi crittograficamente NON SICURI

La sequenza per la cifratura della chiave, cioè per la creazione del KEYSTREAM (file .key):



Lo scambio delle chiavi pubbliche e private: CURVE25519



Un'altra possibile ispirazione al codice sorgente di BABUK: CURVE25519

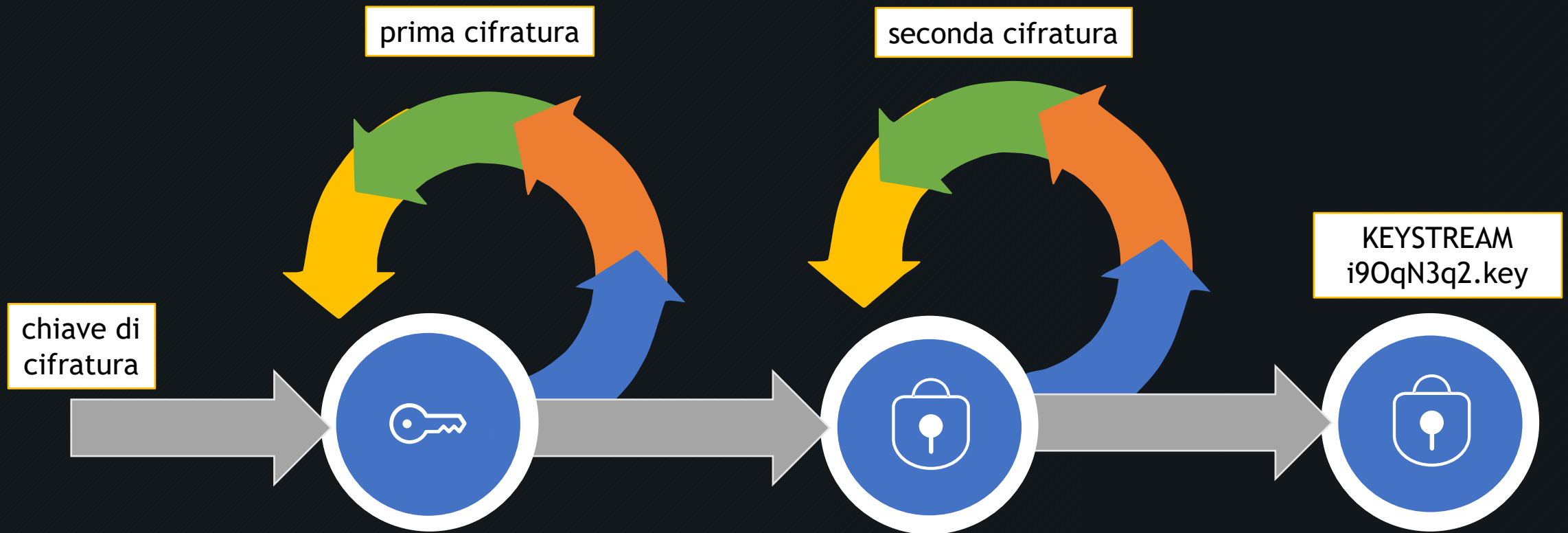
```
void genK() {
    HCRYPTPROV hProv;

    if (!CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_AES, CRYPT_VERIFYCONTEXT) &&
        !CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_AES, CRYPT_VERIFYCONTEXT | CRYPT_NEWKEYSET)) hProv = NULL;
    if (hProv != 0) {
        CryptGenRandom(hProv, 32, k_private);
        k_private[0] &= 248;
        k_private[31] &= 127;
        k_private[31] |= 64;
        curve25519_donna(k_public, k_private, basepoint);

        printf("curve25519 keys generated.\n");
    }
    else {
        printf("Can't initialize HCRYPTPROV, bye!\n");
        ExitProcess(0);
    }
}
```

L'analisi

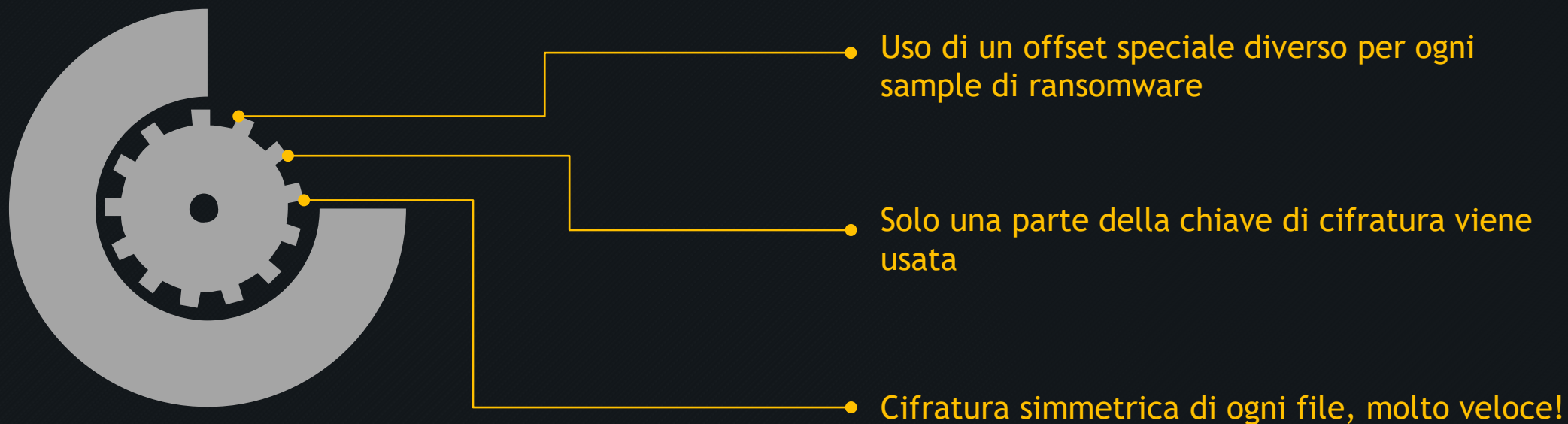
Un perverso concatenamento di algoritmi:



Ricordiamo le operazioni fondamentali di Hive v5:



3. CIFRATURA DEI FILE:



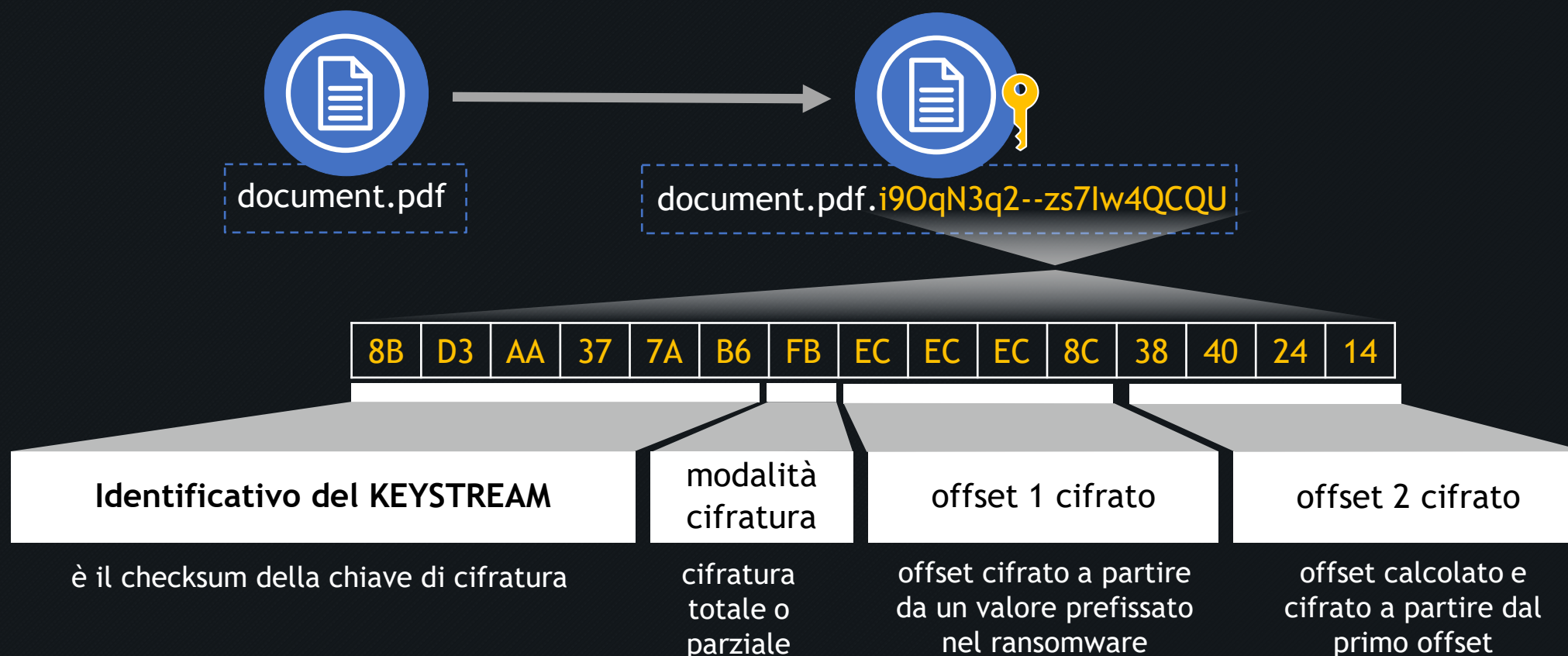
La scelta degli offset. Riprendiamo come esempio la seguente porzione di chiave:

CC	30	94	F8	5C	C0	24	88	50	B4	18	7C	E0	44	A8	0C	70	D4	38	00	64	C8	2C
90	F4	58	BC	20	84	E8	4C	B0	78	DC	40	A4	08	6C	D0	34	98	FC	60	28	8C	F0
54	B8	1C	80	E4	48	AC	10	74	D8	A0	04	68	CC	30	94	F8	5C	C0	24	88	EC	B4
18	7C	E0	44	A8	0C	70	D4	38	9C	00	C8	2C	90	F4	58	BC	20	84	E8	4C	B0	14

1. Gli sviluppatori hanno creato un algoritmo che individua per ogni file due posizioni casuali (offset) all'interno della chiave.
2. La chiave per cifrare un file è l'operazione di XOR bitwise a partire dai due offset scelti:

Offset 1	BC	20	84	E8	4C	B0	78	DC	40	A4	08	6C	...
	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	
Offset 2	CC	30	94	F8	5C	C0	24	88	EC	B4	18	7C	...
	=	=	=	=	=	=	=	=	=	=	=	=	
Chiave di cifratura	70	10	10	10	10	70	5c	54	AC	10	10	10	...

Quando il ransomware cifra un file crea una nuova estensione diversa per ogni file cifrato:

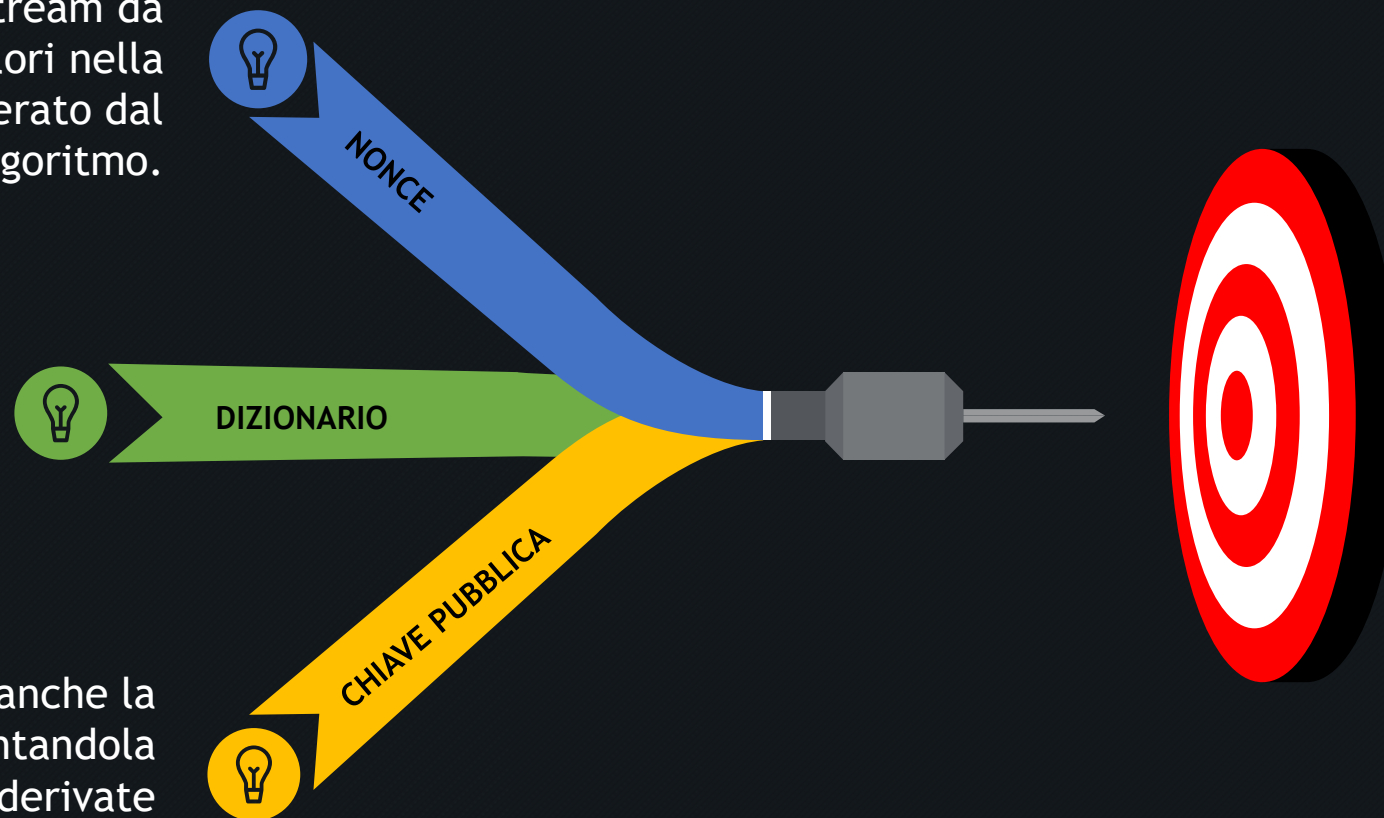


Il keystream decryptor

Il NONCE di 24 byte nel keystream dà un'idea della sequenza di valori nella chiave privata, essendo generato dal medesimo algoritmo.

Un DIZIONARIO di 10 milioni di byte costruito anch'esso con lo stesso algoritmo della chiave privata, fornisce un bacino di byte dove può essere identificata definitivamente.

Nel keystream è riportata anche la chiave pubblica. Confrontandola con le sequenze di byte derivate dal dizionario, si può trovare la relativa chiave privata.



Il keystream decryptor

Il programma serve a decifrare il keystream, sfruttando la predicibilità e la riproducibilità delle chiavi private:

```
Hive ransomware V5 - keystream decryptor PoC
-----

1. dictionary generation using keystream
2. bruteforce existing keystream using computed dictionary
3. check your dictionary using keystream
4. just generate a dictionary series (10)
your move:
```

1

Generazione di massimo 10 dizionari cercando di replicare il FINGERPRINT del NONCE, rallentando l'esecuzione del codice per simulare CPU lente

2

È il vero e proprio cuore del programma: consente di trovare le due chiavi private usando un dizionario che potrebbe contenerle

3

Controllo di un dizionario per vedere se i byte generati contengono il FINGERPRINT del NONCE scritto nel keystream

4

Generazione di 10 dizionari usando solo la frequenza della CPU, senza introdurre ulteriori elementi di rallentamento, al di là dei processi in esecuzione

Fonte:

https://github.com/reecdeep/HiveV5_keystream_decryptor

Il file decryptor

Il programma serve a decifrare i file usando un keystream decifrato con il keystream decryptor

```
Hive ransomware V5 - file decryptor PoC
-----

1. Decrypt a file using decrypted keystream
2. Offset bruteforce
your move:
```

1

Decifrazione dei file usando il keystream decifrato.
È necessario immettere l'offset speciale presente nel sample che ha cifrato i file

2

Dato un file con un header noto (PDF, JPG, PNG, file di Office) bruta il possibile valore dell'offset speciale valido per tutti i file da decifrare

Fonte:

https://github.com/reecdeep/HiveV5_file_decryptor

Conclusioni

- **Perchè questo lavoro?**
Per dare un segnale di presenza costante agli avversari nonostante le difficoltà nell'analisi e reverse engineering del codice.
- **Cosa abbiamo imparato?**
I cyber gruppi criminali emergenti spesso usano codice di altre gang e talvolta, nel tentativo di migliorarli, vi introducono errori che possiamo sfruttare a nostro vantaggio.

Ringraziamenti

Un ringraziamento speciale va a Andrey Zhdanov (@rivitna) per i preziosi consigli durante questi duri mesi di lavoro.

Grazie per l'attenzione!



@reecdeep (Twitter)



[linkedin.com/in/riccardodp](https://www.linkedin.com/in/riccardodp)