Bisonal: 10 years of play

blog.talosintelligence.com/2020/03/bisonal-10-years-of-play.html

By Warren Mercer, Paul Rascagneres and Vitor Ventura.

Update 06/03/20: added samples from 2020.

Executive summary

- Security researchers detected and exposed the Bisonal malware over the past 10 years. But the Tonto team, the threat actor behind it, didn't stop.
- The victimology didn't change over time, either. Japanese, South Korean and Russian organizations were the prime targets for this threat actor.
- The malware evolved to lower its detection ratio and improve the initial vector success rate.

What's new?

Bisonal is a remote access trojan (RAT) that's part of the Tonto Team arsenal. The peculiarity of the RAT is that it's been in use for more than 10 years — this is an uncommon and long period for malware. Over the years, it has evolved and adapted mechanisms to avoid detection while keeping the core of its RAT the same. We identified specific functions here for more than six years.

How did it work?

Bisonal used multiple lure documents to entice their victims to open and then be infected with Bisonal malware. This group has continued its operations for over a decade and they continue to evolve their malware to avoid detection. Bisonal primarily used spear phishing to obtain a foothold within their victims' networks. Their campaigns had very specific targets which would suggest their end game was more around operational intelligence gathering and espionage.

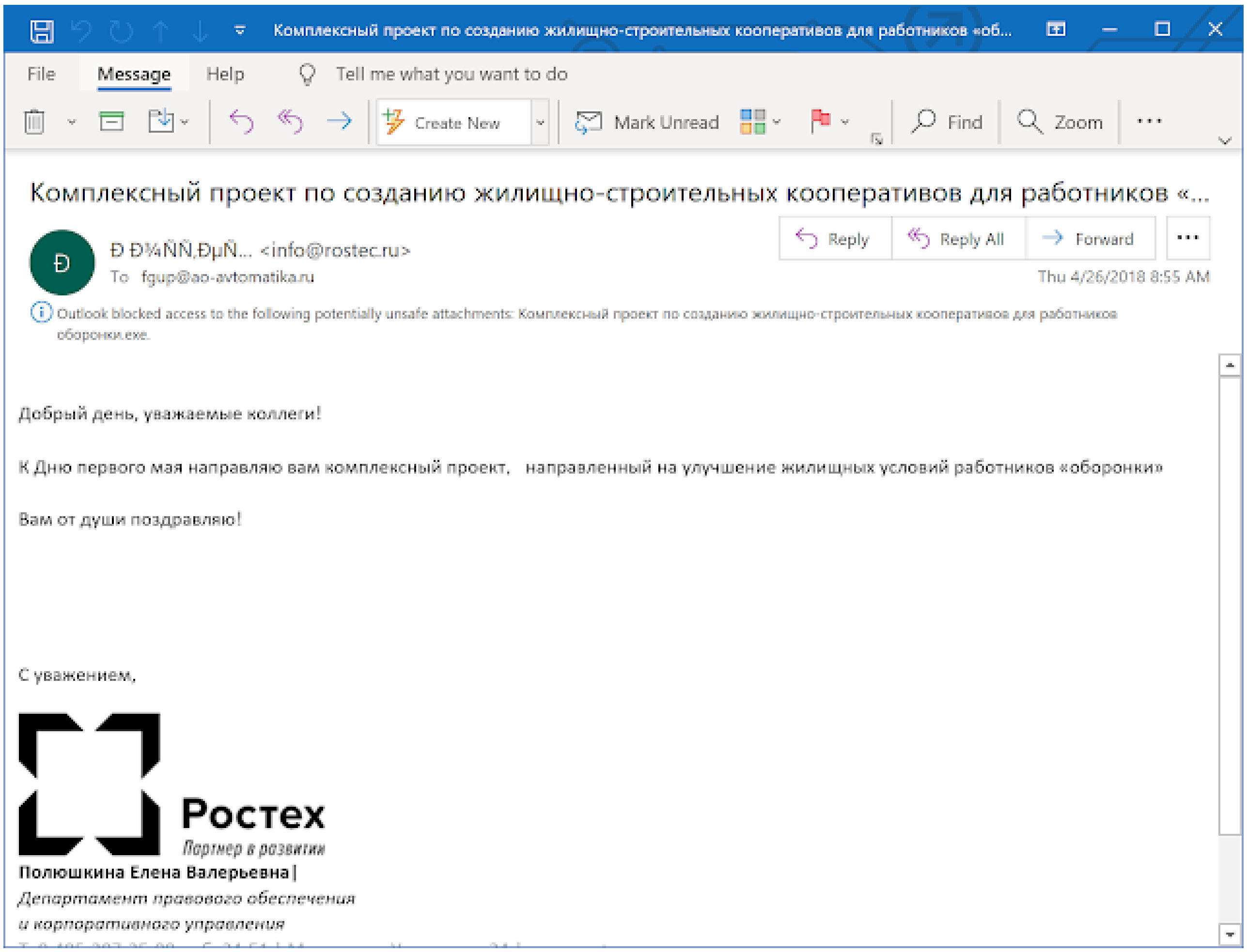
So what?

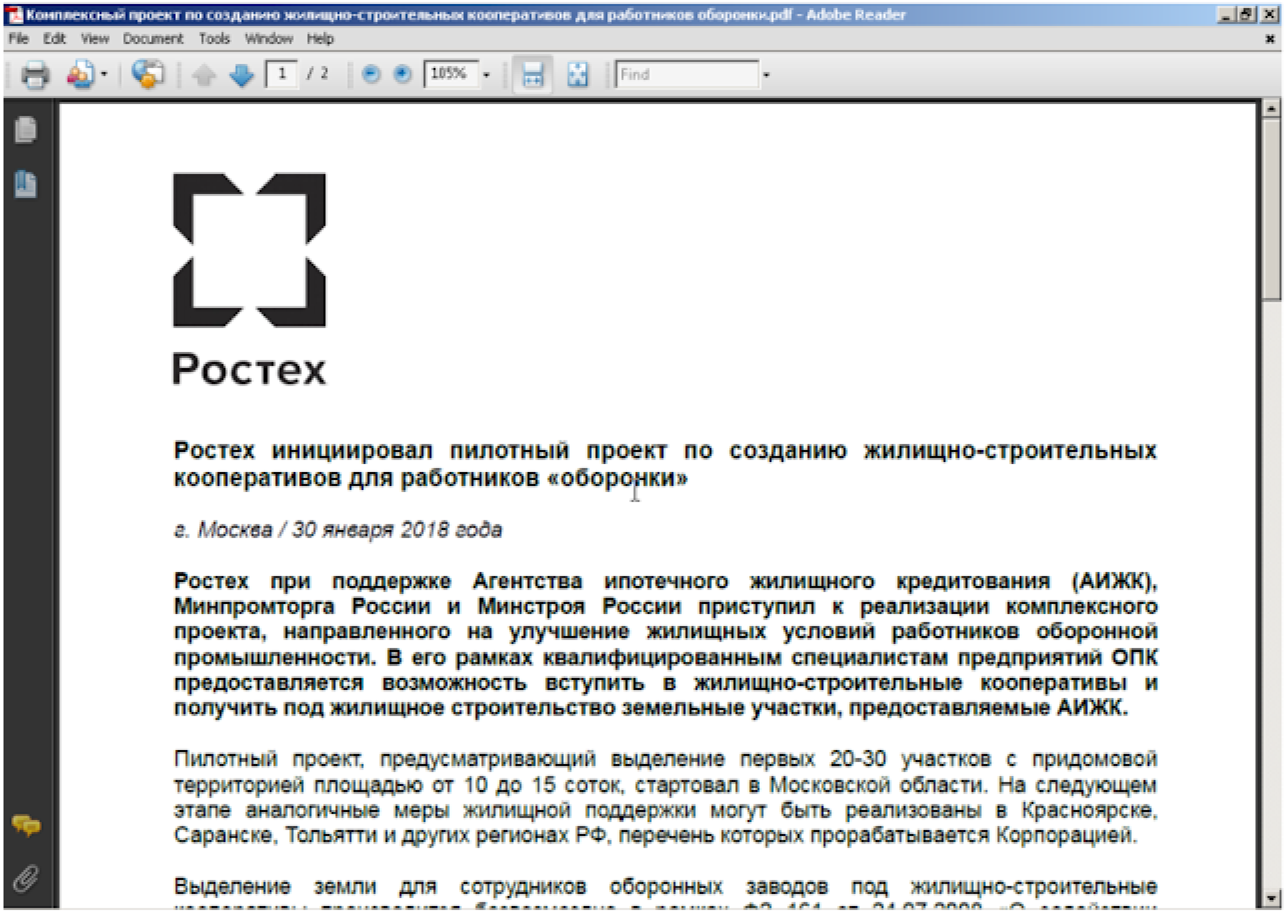
This is an extremely experienced group likely to keep their activities even after exposure, even if we identified mistakes and bad copy/paste, they are doing this job for more than 10 years. We think that exposing this malware, explaining the behavior and the campaigns where Bisonal was used is important to protect the potential future targets. The targets to this point are located in the public and private sectors with a focus on Russia, Japan and South Korea. We recommend the entities located in this area to prepare for this malware and actor and implement detections based on the technical details provided in this article.

Victimology and campaigns

From our analysis and the intelligence shared by the community throughout the last decade of activities of Bisonal, we can conclude that the actor behind this malware is specifically targeted at the South East Asian region, namely Japan and Korea with another significant focus on Russian-speaking victims.

We identified a couple of decoy documents pointing to the victims. During the Heartbeat campaign documented in 2012 by Trend Micro, dating back to 2009, the attacker used Hangul Word Processor (HWP) decoy documents. This file format is mainly used in South Korea. The report mentioned political parties, media outfits, a national policy research institute, a military branch of South Korean armed forces, a small business sector organization and branches of the South Korean government. Later in 2018, Unit 42 released a Bisonal paper where we can see a spear-phishing campaign in Russian and a decoy document alleged to be from Rostec, a Russian state-owned holding conglomerate headquartered in Moscow.





Комплексный проект по созданию жилищно-строительных кооперативов для работников оборонки.pdf - Adobe Reader

File Edit View Document Tools Window Help

Ростех

Ростех инициировал пилотный проект по созданию жилищно-строительных кооперативов для работников «оборонки»

г. Москва / 30 января 2018 года

Ростех при поддержке Агентства ипотечного жилищного кредитования (АИЖК), Минпромторга России и Минстроя России приступил к реализации комплексного проекта, направленного на улучшение жилищных условий работников оборонной промышленности. В его рамках квалифицированным специалистам предприятий ОПК предоставляется возможность вступить в жилищно-строительные кооперативы и получить под жилищное строительство земельные участки, предоставляемые АИЖК.

Пилотный проект, предусматривающий выделение первых 20-30 участков с придомовой территорией площадью от 10 до 15 соток, стартовал в Московской области. На следующем этапе аналогичные меры жилищной поддержки могут быть реализованы в Красноярске, Саранске, Тольятти и других регионах РФ, перечень которых прорабатывается Корпорацией.

Выделение земли для сотрудников оборонных заводов под жилищно-строительные

Finally, in 2018, Ahnlab released a paper about "Operation Bitter Biscuit" where Bisonal was used against Korean and Japanese entities. India is also mentioned, but it was by another malware named "Bioazih" by Ahnlab. In this paper, the editor mentions targets such as manufacturers, defense industry and government.

Additionally, we can provide additional decoy documents. For example, a Korean document used in September 2014 where the title was "Contact member and counselor of the Agriculture, Forestry, Livestock, Food and Marine Fisheries Committee:"

농림축산식품해양수산위원회 위원 및 보좌진 연락처

구분	위원명	일반 (784)	구내 (788)	FAX (788)	보 좌 진	이메일	호실
새정치 민주연합	◎김우남 010-3691-8070	1368	2725	3690	김병찬 010-3169-7000 김혜선 010-3292-6284 박혜림 010-4135-1474 김미나 010-9437-8762	김미숙 nina61@naver.com 김병찬 rla1342@nate.com 김혜선 yesun1019@hanmail.net 박혜림 human8081@naver.com 이동주 djlee75@hanmail.net 이현정 aramal1031@jejunu.ac.kr 김미나 kimmina1989@nate.com	
	▲안효대 010-3834-9271	2371	2365	0254	전정배 010-8943-1457 김병하 010-2995-6451 박진운 010-9267-2001 정홍석 010-3639-4035 박수지 010-5203-1547 이용한 010-6413-2770	대 표 ahd823@hanmail.net	823
	경 대 수 010-5351-4109	3977	2009	0110	김종춘 010-5005-3167 안중우 010-6571-2587 강민경 010-4138-9968	의원님 kyungds@na.go.kr 의원실 eskfo@nate.com 김종춘 justice@na.go.kr 안중우 ajw2594@assembly.go.kr 조성재 bluyss@empal.com	941
	김 무 성 010-9038-5274	5274	2693	0316	권오훈 010-4744-8811	moosung4u@naver.com resistx@nate.com	706
	김 종 태 010-3651-6363	3190	2149	0518	김홍태 010-2120-2780 최성용 010-4769-7450 이경식 010-2075-0000 이현영 010-4429-1101 김태현 010-8793-5992	김홍태 hogtae0924@hanmail.net 최성용 tiger@na.go.kr 정문기 jmg5812@hanmail.net 이경식 fop777@na.go.kr 이현영 say0024@naver.com 김태현 nakth@na.go.kr 김용석 skql7121@naver.com	452
새 ㄴ	안 덕 수 010-5238-1269	9640	2490	0250	송기영 010-2784-8898 이상엽 010-3899-1416 박상희 010-4517-1105 이주영 010-9727-6246	송기영 phisky@hanmail.net 이상엽 950165@hanmail.net 박상희 77babsang@hanmail.net 이주영 k9346245@naver.com	341

Or a Russian document about the CIPR Digital conference used in April 2018. This is an application document that has been used to provide a decoy to the Bisonal malware. This conference has some high-ranking government and business attendees.

ЦИПР

ЦИФРОВАЯ ИНДУСТРИЯ ПРОМЫШЛЕННОЙ РОССИИ

ФОРМА ПОДАЧИ ЗАЯВКИ НА ВЫСТУПЛЕНИЕ

Инструкция по оформлению заявки на выступление в рамках Конференции ЦИПР:

1. Необходимо заполнить регистрационную форму докладчика до 27 апреля 2018 года.
С 27 апреля 2018 года по 25 мая 2018 года стать докладчиком Конференции возможно только по личному приглашению членов оргкомитета ЦИПР.

2. Необходимо получить одобрение темы модератором. Рассмотрение заявки занимает не более 15 дней. После чего Ваш доклад может быть включен в официальную программу, а информация о Вас появится в разделе Участники официального сайта конференции cipr.ru

3. Тема Вашего выступления должна соответствовать ключевым темам Конференции ЦИПР-2018. Длительность выступления – от 3 до 5 мин.

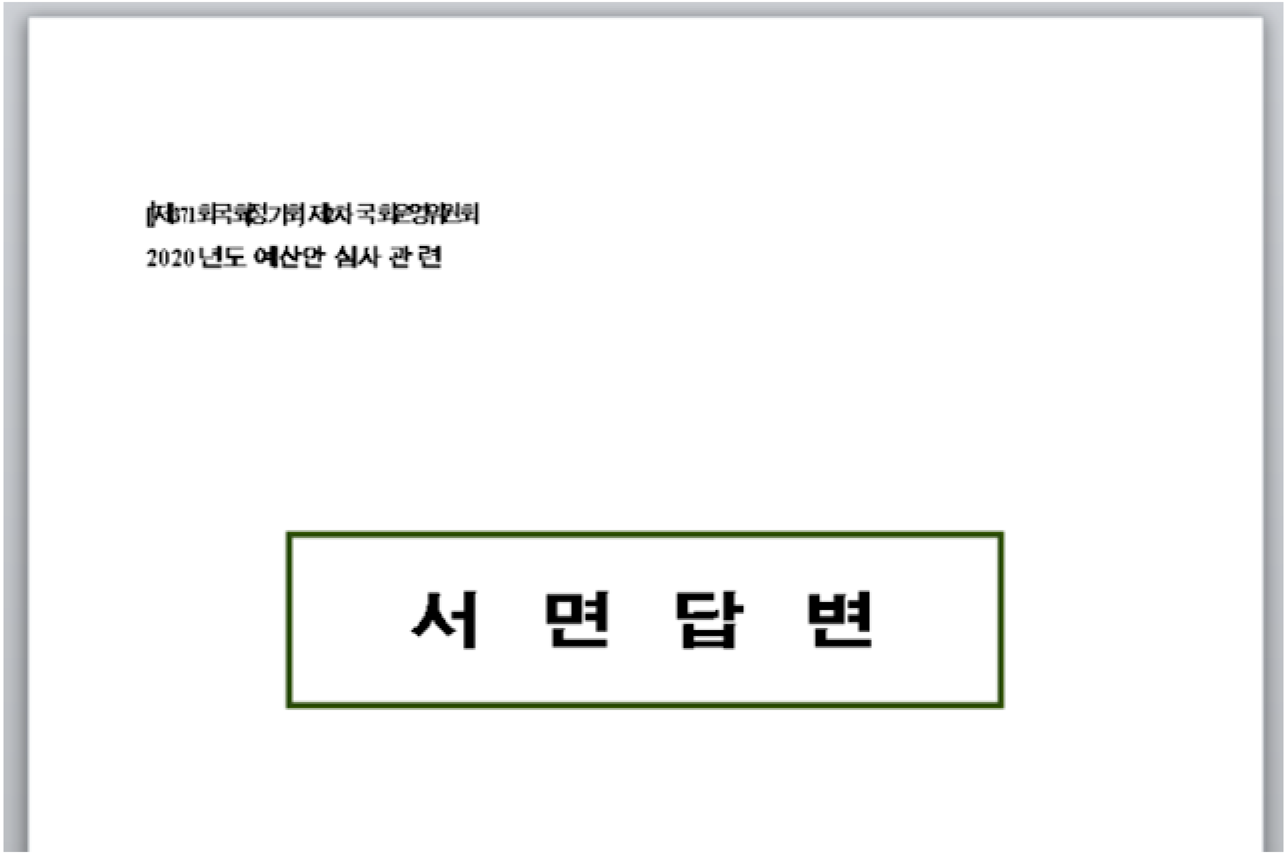
Если вы согласны с условиями выше, просим Вас заполнить форму:

Фамилия	
Имя	
Отчество	
Должность	
Название организации	
Ученая степень / регалии	

In 2019, a Russian RTF document — судалгаа.doc (research.doc) — was used with an exploit to drop the winhelp.wll file, which contains Bisonal.

д/д	Улс	тоо	Тайлбар
1.	Бангладеш		
2.	Бельги		
3.	Бенин		
4.	Бутан		
5.	Болив		
6.	Босни Герцоговин		
7.	Бразил		
8.	Буркина Фасо		
9.	Камерун		
10.	Канад		
11.	БНХАУ		
12.	Чех		
13.	Египет		
14.	Франц		
15.	Гана		
16.	Гуатимал		
17.	Энэтхэг		
18.	Индонез		
19.	Ирланд		

Last year, we also identified multiple Korean decoy documents using similar RTF exploits to deliver Bisonal, namely ☆2020년도 예산안 운영위 서면질의 답변서_발간(1).doc (State Council Candidate (Minister of Justice Chumiae) Personnel Hearing Execution Plan (1) .doc) and 국무위원후보자(법무부장관 추미애) 인사청문회 실시계획서(1).doc (Written Inquiry from the 2020 Budget Operation Committee (Published) (1) .doc) which are both alleged government documents.



Based on our research and the released paper mentioned above, the Bisonal malware is part of the Tonto Team arsenal. Tonto Team was mentioned in the media in 2017 as one of the actors who targeted South Korea, when the country announced it would deploy a Terminal High-Altitude Air Defense (THAAD) in response to North Korean missile tests. At this time, researchers connected the Tonto Team to China.

10 years of evolution

Introduction

The first variant of Bisonal publicly released went by the name of "HeartBeat." At the end of 2019, the actor changed their TTP and started using the Microsoft Office extension (.wll) to execute the Bisonal payload. Based on this recent change, we decided to dive into the 10 years of evolution of Bisonal. To do so, we analysed more than 50 different samples and focused on the changes that appear during the years of usage.

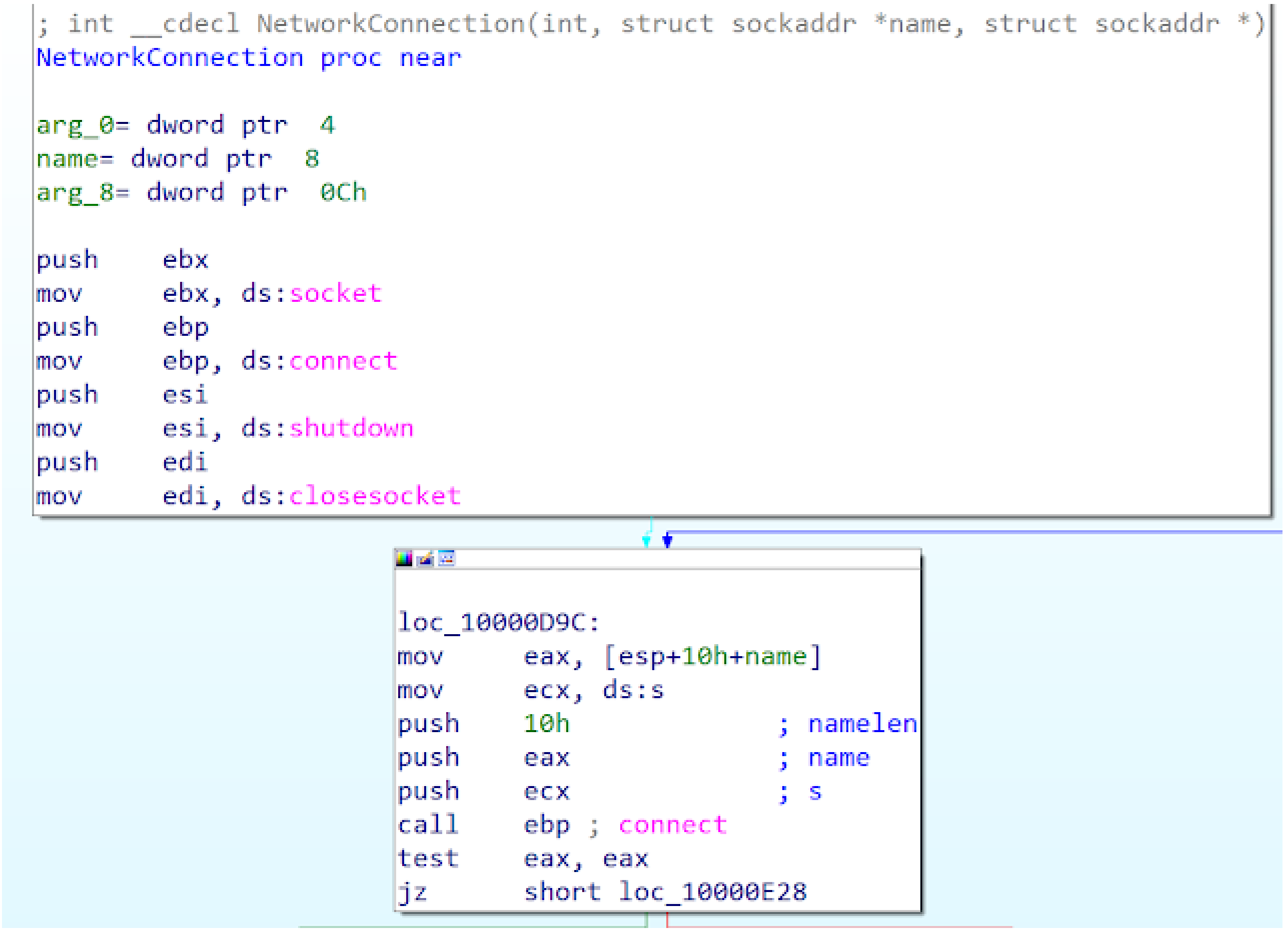
2010: the birth

The oldest version of Bisonal we identified was compiled on Dec. 24, 2010. This version is the simplest we identified. The attacker created a Windows library (.dll) designed as a Windows service (ServiceMain() entry point). When executed, the malware uses the Windows API to communicate with the Service Control Manager (SCM) and finally execute a thread. This thread contains the code of the malware.

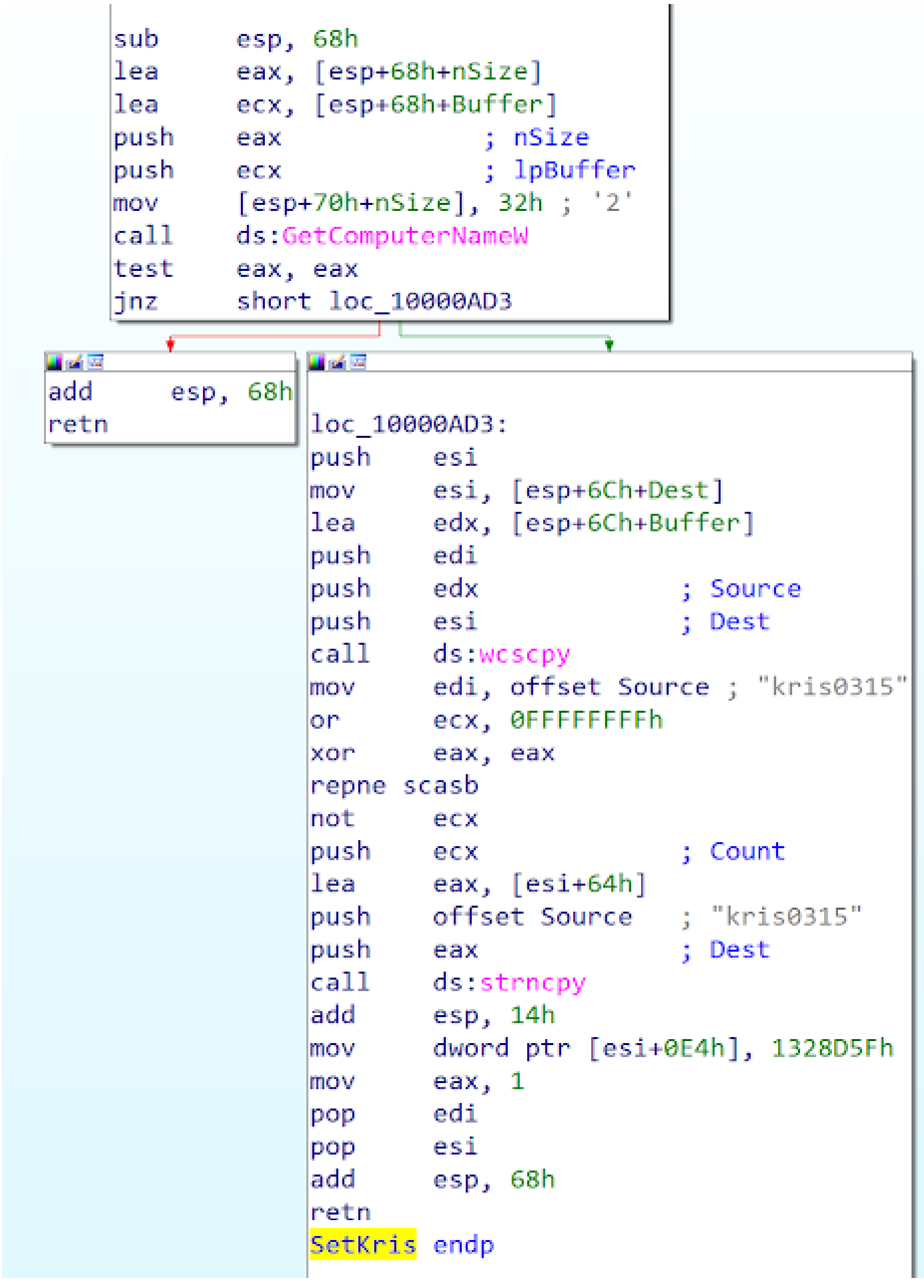
The C2 server of this first Bisonal variant is young03[.]myfw[.]us (port 8888). We can notice the usage of a dynamic DNS service. This is a Bisonal pattern. Even the newest version we identified used this kind of service. The domain name was not obfuscated:

```
lea     eax, [esp+108h+cp]
push    eax                 ; int
push    offset MultiByteStr ; "young03.myfw.us"
call    GetHostByName_API
mov     ecx, 1Fh
xor     eax, eax
lea     edi, [esp+110h+var_7F]
mov     [esp+110h+var_80], 0
rep stosd
stosw
lea     ecx, [esp+110h+var_80]
push    ecx                 ; int
push    offset a127001      ; "127.0.0.1"
stosb
call    GetHostByName_API
mov     esi, ds:inet_addr
add     esp, 10h
```

The IP address is a rollback if the first C2 server is down. In this campaign, the rollback was not used as it is configured to localhost. The communication to the C2 server is performed by using raw sockets:



The first action of the malware is to send the hostname of the infected system and the "kris0315" string. The sent data is not encrypted or obfuscated. We assume the string is an identifier:



The malware supports only three commands:

- Command execution: The execution is performed by the ShellExecuteW() API
- Listing the running processes
- Cleaning the malware: The malware first removes the registry key of the service and removes the library. As the library is currently running, the deletion cannot be performed immediately. The developer decided to use MoveFileEx() API with the MOVE_DELAY_UNTIL_REBOOT to remove the file at the reboot.

The malware contains the Bisonal string. It is interesting to notice the string is not used but is still visible:

Address	Length	Type	String
.text:10000390	00000008	C	bisonal
.text:10000398	00000010	C	young03.myfw.us
.text:1000041C	0000000A	C	127.0.0.1
.text:100004A0	00000009	C	kris0315
.text:10000578	0000000E	C	SocketError:

The sample was used in the HeartBeat campaign mentioned above.

Sha256: ba0bcf05aaefa17fbf99b1b2fa924edbd761a20329c59fb73adbaae2a68d2307
C2 server: young03[.]myfw[.]us

2011: obfuscation my darling & more espionage capabilities

2011 March: commect()

We identified a sample from March 18, 2011. The sample is really similar to the variant from 2010. We can notice that the developers wanted to hide some API usage. They use the LoadLibrary() API followed by GetProcAdress(). But they obfuscated the function name strings by splitting it in two. Here is an example:

```
.data:100024C4 ; CHAR LibFileName[]
.data:100024C4 LibFileName     db 'Ws2_32.dll',0
.data:100024CF                 align 10h
.data:100024D0 Connect_Part1   dd 'mmoc'
.data:100024D4 Connect_Part2   dd 'tce'
```

Once the two strings are concatenated and with the little-endian, the string becomes "commect." After the malware replaces the "m" by "n:"

```
sub     esp, 0Ch
mov     eax, Connect_Part1
mov     ecx, Connect_Part2
mov     dword ptr [esp+0Ch+ProcName], eax
push    esi
mov     al, 6Eh ; 'n'
push    offset LibFileName ; "Ws2_32.dll"
mov     [esp+14h+var_8], ecx
mov     [esp+14h+var_4], 0
mov     [esp+14h+ProcName+3], al
mov     [esp+14h+ProcName+2], al
call    ds:LoadLibraryA
lea     edx, [esp+10h+ProcName]
mov     esi, eax
push    edx             ; lpProcName
push    esi             ; hModule
call    ds:GetProcAddress
test    eax, eax
jz      short loc_10000409
```

They use this trick for a couple of other API such as CreateThread(), CreatePipe(), PeekNamedPipe(), CreateProcessA(), CreateToolhelp32Snapshot(), ReadFile(), WriteFile() and, finally, the string "cmd.exe."

The attacker also implemented a new order: execution of a command by using named pipe to get the output of the executed command. The attackers execute cmd.exe, followed by the command to be executed. An interesting point is the adding of a charset on each executed command:

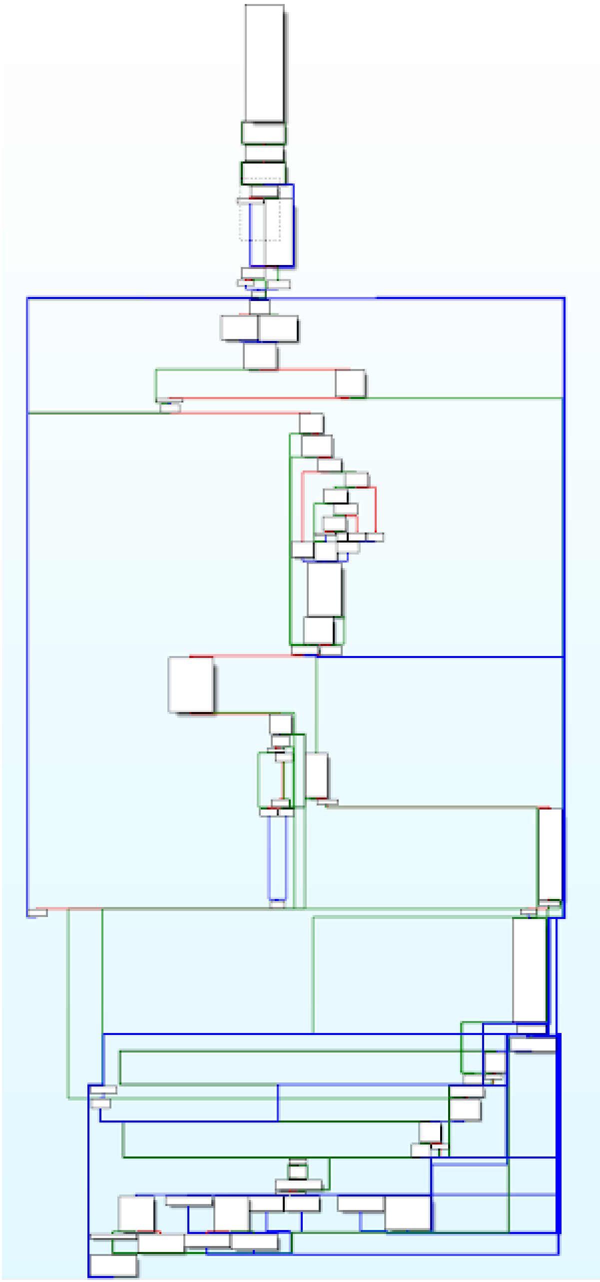
```
mov     eax, [esp+690h+hFile]
lea     edx, [esp+690h+NumberOfBytesWritten]
push    0                   ; lpOverlapped
push    edx                 ; lpNumberOfBytesWritten
push    0Ah                 ; nNumberOfBytesToWrite
push    offset aChcp1251 ; "CHCP 1251\n"
push    eax                 ; hFile
mov     [esp+6A4h+var_670], 1
call    ds:WriteFile
```

This charset is designed to cover languages that use Cyrillic script such as Russian, Bulgarian and Serbian. This hardcoded string could be an indicator concerning the targets of this malware.

sha256 : bb61cc261508d36d97d589d8eb48aaba10f5707d223ab5d5e34d98947c2f72af
C2 server: kissyou01[.]myfw[.]us

2011 September: The big changes

The developer decided to remove the MFC library and put almost all the code in a unique function. The number of functions is divided by three. Here is the main thread graph flow:



Additionally, the string such as the domain names of the URLs is encoded by using the XOR algorithm (0x1f for example). The network communication is also obfuscated with a XOR (0x28).

On the version, the attacker supports the proxy server. It was a limitation of the previous variants. If the target would have a proxy, the malware would not be able to communicate outside. The attacker retrieves the proxy configuration in the registry:

```
loc_7100159C:
mov     edx, [esp+0E4h+phkResult]
mov     ebx, ds:RegQueryValueExA
lea     eax, [esp+0E4h+cbData]
lea     ecx, [esp+0E4h+Type]
push    eax                 ; lpcbData
push    0                   ; lpData
push    ecx                 ; lpType
push    0                   ; lpReserved
push    offset ValueName ; "ProxyServer"
push    edx                 ; hKey
call    ebx ; RegQueryValueExA
mov     eax, [esp+0E4h+cbData]
push    eax                 ; unsigned int
call    ??2@YAPAXI@Z        ; operator new(uint)
mov     ecx, [esp+0E8h+cbData]
mov     esi, eax
mov     edx, ecx
xor     eax, eax
mov     edi, esi
add     esp, 4
shr     ecx, 2
rep stosd
mov     ecx, edx
and     ecx, 3
rep stosb
mov     edx, [esp+0E4h+phkResult]
lea     eax, [esp+0E4h+cbData]
push    eax                 ; lpcbData
lea     ecx, [esp+0E8h+Type]
push    esi                 ; lpData
push    ecx                 ; lpType
push    0                   ; lpReserved
push    offset ValueName ; "ProxyServer"
push    edx                 ; hKey
call    ebx ; RegQueryValueExA
test    eax, eax
jz      short loc_7100160B
```

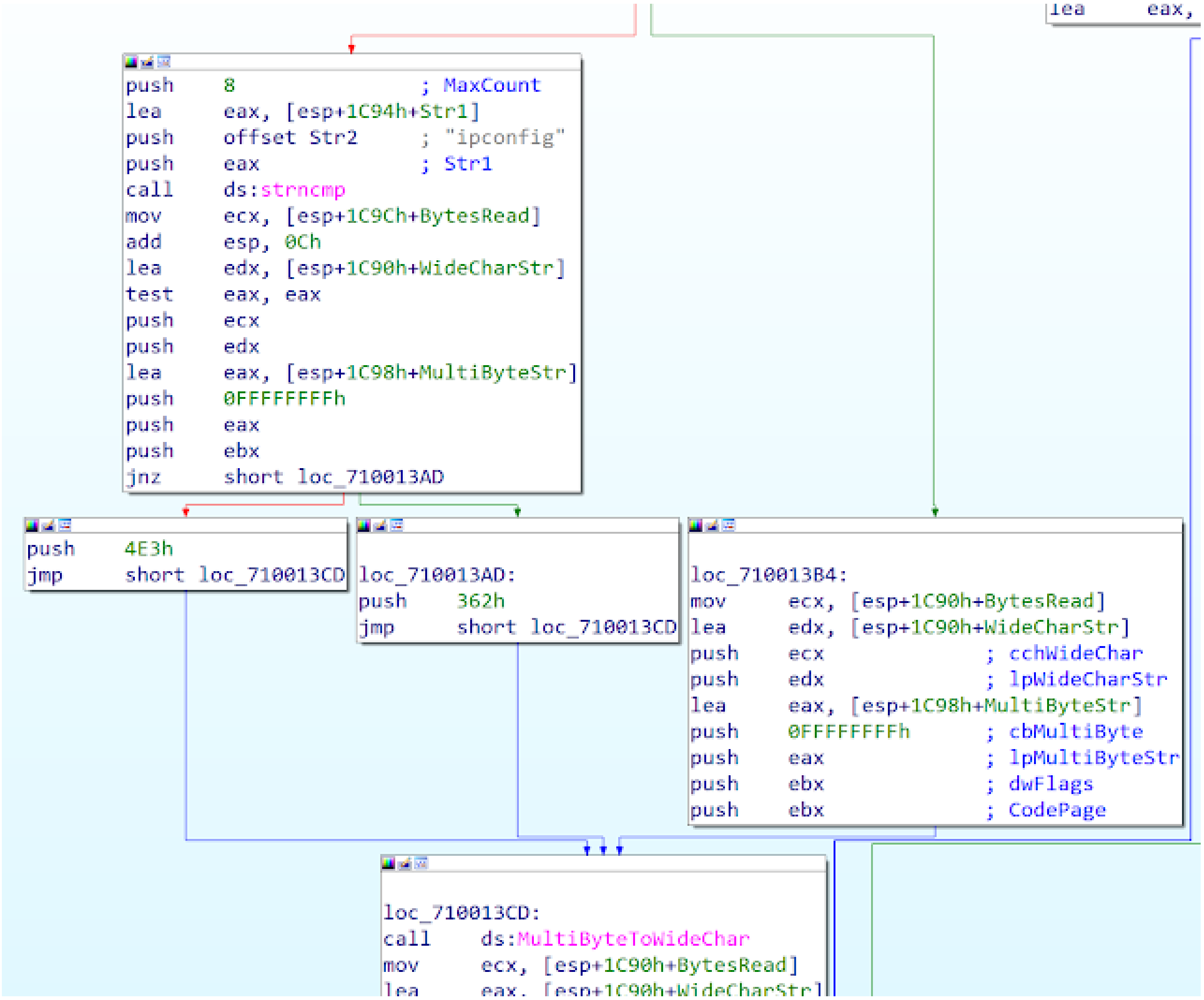
The network communication is divided in two parts. The first part uses the Microsoft Windows Wininet library. The purpose is to send reconnaissance information to the attackers. The data is sent to the server via InternetOpenA() and InternetOpenURLA(). The C2 server of the analysed sample is hxxp://fund[.]cmc[.]or[.]kr/UploadFile/fame/x/o0.asp. The malware sent to the operator the following information: the campaign ID (named Flag by the developer), the hostname of the compromised system, the IP address, the OS version, the proxy server of the system and if the system is running on VMware. To get this information, the attacker the VMXh-Magic-Value (0x0a). The second part of the communication is dedicated to the orders and the exfiltration. This part is similar to the previous samples: raw sockets usage.

The features of the malware are the same as previously with new capabilities such as file creation and removal.

The author removed the malware cleaning feature and implements two others features: the developer adds PostThreadMessageW() to send message inside the thread and in the previous version the developer used TerminalProcess() API to stop the process executed via the named pipes, in the version the developer append the "exit\r\b" string to the executed command in order to exit properly:

```
xor     eax, eax
repne scasb
mov     edx, [esp+1C98h+hWritePipe]
not     ecx
dec     ecx
push    ecx                 ; nNumberOfBytesToWrite
lea     ecx, [esp+1C9Ch+Str1]
push    ecx                 ; lpBuffer
push    edx                 ; hFile
call    ds:WriteFile
push    1F4h                ; dwMilliseconds
call    ebp ; Sleep
mov     esi, offset aExit ; "exit\r\n"
lea     eax, [esp+1C90h+Str1]
```

Another interesting change is the fact they don't use CHCP command anymore to force the charset but use code page. You can see in the screenshot 0x4E3 (1251 - Cyrillic Russian) and 0x362 (866 - DOS Cyrillic Russian):



Sha256: 43606116e03672d5c2bca7d072caa573d3fc2463795427d6f5abfa25403bd280
C2 for the orders: dnsdns1[.]PassAs[.]us
C2 URL for reconnaissance: hxxp://fund[.]cmc[.]or[.]kr/UploadFile/fame/x/o0.asp

2011 October: oops where is my cleaning function?

In October 2011, the attacker re-implements the cleaning function.

```
loc_7100229C:            ; hSCObject
push    esi
mov     esi, ds:CloseServiceHandle
call    esi ; CloseServiceHandle
push    edi              ; hSCObject
call    esi ; CloseServiceHandle
mov     ecx, 7
mov     esi, offset aCWindowsSystem ; "c:\\windows\\system32\\rudll.dll"
lea     edi, [ebp+ExistingFileName]
rep movsd
movsw
mov     ecx, 39h ; '9'
xor     eax, eax
lea     edi, [ebp+var_106]
rep stosd
stosw
push    4                ; dwFlags
push    0                ; lpNewFileName
lea     eax, [ebp+ExistingFileName]
push    eax              ; lpExistingFileName
call    ds:MoveFileExA
mov     eax, 1
mov     ecx, [ebp+ms_exc.registration.Next]
mov     large fs:0, ecx
pop     edi
pop     esi
pop     ebx
mov     esp, ebp
pop     ebp
retn
; } // starts at 710021B0
MalwareCleaning endp
```

In this implementation, the developer first uses the Windows service management API in order to remove the service (instead of removing directly the registry key as he did previously) and, finally, remove the file with the same API as previously (MoveFileExA()).

Sha256:43459f5117bee7b49f2cee7ce934471e01fb2aa2856f230943460e14e19183a6
C2 for the orders: jennifer998[.]lookin[.]at
C2 for rollback: 196[.]44[.]49[.]154
C2 URL for reconnaissance: hxxp://fund[.]cmc[.]or[.]kr/UploadFile/fame/x/o0.asp

2011 December: Not a service anymore

The new variant from December 2011 is not a service anymore but a simple library (.dll). The library is executed via a launcher (conime.exe) and the persistence mechanism is not a service anymore but a registry key (CurrentVersion\\Run\\task).

The malware is lighter than the previous version but includes more espionage features such as file exfiltration, file listing, driver listing, process-killing, file removing. The other features are the same as previously.

It is interesting to note that the obfuscated reconnaissance is still hard-coded in the binary but it is not used anymore. The code used for the reconnaissance was removed but the developer forgot the IP variable.

Sha256: 915ad316cfd48755a9e429dd5aacbee266aca9c454e9cf9507c81b30cc4222e5
C2 for the orders: v3net[.]rr[.]nu
C2 for rollback: faceto[.]UglyAs[.]com
C2 URL for reconnaissance: hxxp://fund[.]cmc[.]or[.]kr/UploadFile/fame/x/mh/o.asp

Hardcoded identifiers

In this version, we identify hard coded identifiers. We assume these IDs are campaign or target ID. Here is a list of IDs:

- 1031
- jp0201
- jp-serv
- mhi
- m1213
- classnk
- 95mhi
- nscsvc

In the next version, a campaign ID will be also used. The ID we believe is in reference to Japan targets. We believe these targets to sit within both the public and private sectors and they are specifically targeted to further enhance the attacker's capabilities through espionage.

2012: File format year

February: Let's hide my code in an almost legit library

In February 2012, the developer tried to hide the malicious code in the middle of a legit library. The malicious library was named msacm32.dll and contains the same exports as a legit library from Microsoft Windows named msacm.dll. Here is the export of the malicious library with the same name than the real one:

XRegThunkEntry	100023D0	1
acmDriverAddA	100023E0	2
acmDriverAddW	100023F0	3
acmDriverClose	10002400	4
acmDriverDetailsA	10002410	5
acmDriverDetailsW	10002420	6
acmDriverEnum	10002430	7
acmDriverID	10002440	8
acmDriverMessage	10002450	9
acmDriverOpen	10002460	10
acmDriverPriority	10002470	11
acmDriverRemove	10002480	12
acmFilterChooseA	10002490	13
acmFilterChooseW	100024A0	14
acmFilterDetailsA	100024B0	15
acmFilterDetailsW	100024C0	16
acmFilterEnumA	100024D0	17
acmFilterEnumW	100024E0	18
acmFilterTagDetailsA	100024F0	19
acmFilterTagDetailsW	10002500	20
acmFilterTagEnumA	10002510	21
acmFilterTagEnumW	10002520	22
acmFormatChooseA	10002530	23
acmFormatChooseW	10002540	24
acmFormatDetailsA	10002550	25
acmFormatDetailsW	10002560	26
acmFormatEnumA	10002570	27
acmFormatEnumW	10002580	28
acmFormatSuggest	10002590	29
acmFormatTagDetailsA	100025A0	30
acmFormatTagDetailsW	100025B0	31

As previously the hard-coded C2 for reconnaissance variable is here. Without being used.

Sha256: 6f8bbea18965b21dc8b9163a5d5205e2c5e84d6a4f8629b06abe73b11a809cca
C2 for the orders: since[.]qpoe[.]com
C2 for rollback: applejp[.]myfw[.]us
C2 URL for reconnaissance: hxxp://fund[.]cmc[.]or[.]kr/UploadFile/fame/x/o0.asp

2012 May & December: I miss services

In May and December 2012, the developers modified the .dll to come back to a Windows service.

As previously described, the hardcoded C2 for reconnaissance variable is here. Without being used.

Sha256: b75c986cf63e0b5c201da228675da4eff53c701746853dfba6747bd287bdbb1d
C2 for the orders: since[.]qpoe[.]com
C2 for rollback: 69[.]197[.]149[.]98
C2 URL for reconnaissance: hxxp://fund[.]cmc[.]or[.]kr/UploadFile/fame/x/o0.asp

Sha256: 979d4e6665ddd4c515f916ad9e9efd9eca7550290507848c52cf824dfbd72a7e
C2 for the orders: usababa[.]myfw[.]us
C2 for rollback: indbaba[.]myfw[.]us
C2 URL for reconnaissance: hxxp://indbababababa[.]dns94[.]com/o.asp

2012 October: Standalone PE

In October 2012, the attackers used an .exe. The attacker chose a standalone PE.

As previously the hard coded C2 for reconnaissance variable is here. without being used.

Sha256: 6f4a1b423c3936969717b1cfb25437ae8d779c095f158e3fded94aba6b6171ad
C2 for the orders: mycount[.]MrsLove[.]com
C2 for rollback: mycount[.]MrsLove[.]com
C2 URL for reconnaissance: hxxp://fund[.]cmc[.]or[.]kr/UploadFile/fame/x/o0.asp

2013: RIP

We did not identify any Bisonal samples used in 2013. The first explanation could be that it was used so much that it stays under our radar. The second explanation could be a publication from Trend Micro on January 3, 2013. In the publication, the editor described a campaign where Bisonal was used. Maybe the actor decided to stop using Bisonal?

2014: The rebirth

Packer

For the first time, the Bisonal developers decided to use a packer: MPRESS. The Bisonal string also disappears from the binary however the workflow of the malware stays the same and some features are copy/pasted from the previous Bisonal variant.

Obfuscation

The domain and the port number are obfuscated but it is not a simple XOR anymore. The developers implemented its own byte manipulation algorithm. The developer also implemented an obfuscation concerning OS detection. The OS version string is not stored as a string anymore but as bytes:

```
loc_4024CC:
mov     [esp+50Ch+var_474], 56h ; 'V'
mov     [esp+50Ch+var_472], 69h ; 'i'
mov     [esp+50Ch+var_470], 73h ; 's'
mov     [esp+50Ch+var_46E], 74h ; 't'
mov     [esp+50Ch+var_46C], 61h ; 'a'
jmp     loc_40262F
```

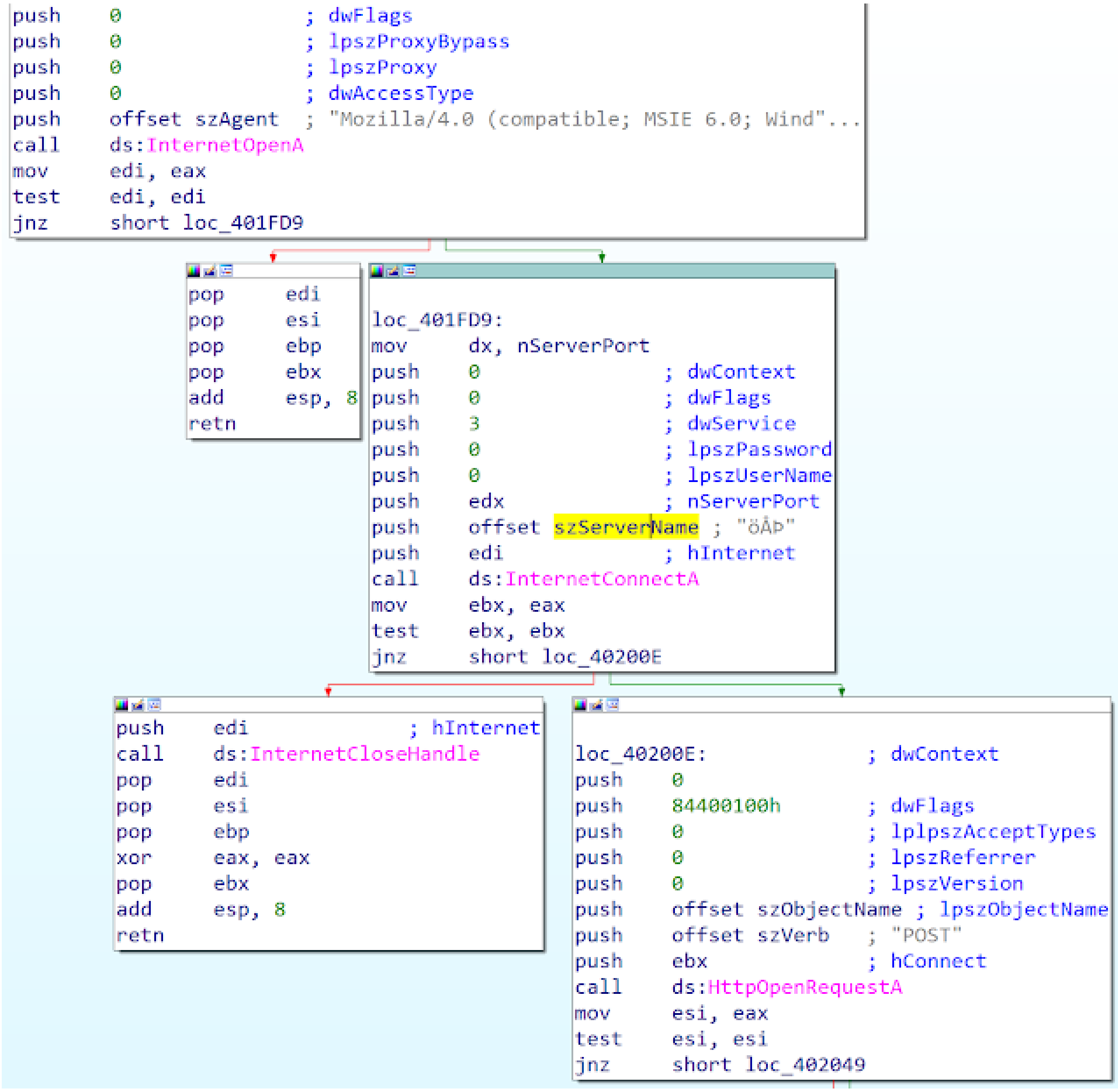
```
mov     [esp+50Ch+var_474], 37h ; '7'
jmp     loc_40262F
```

It is interesting to note that a few samples from 2014 do not use the obfuscation described above.

Malware core

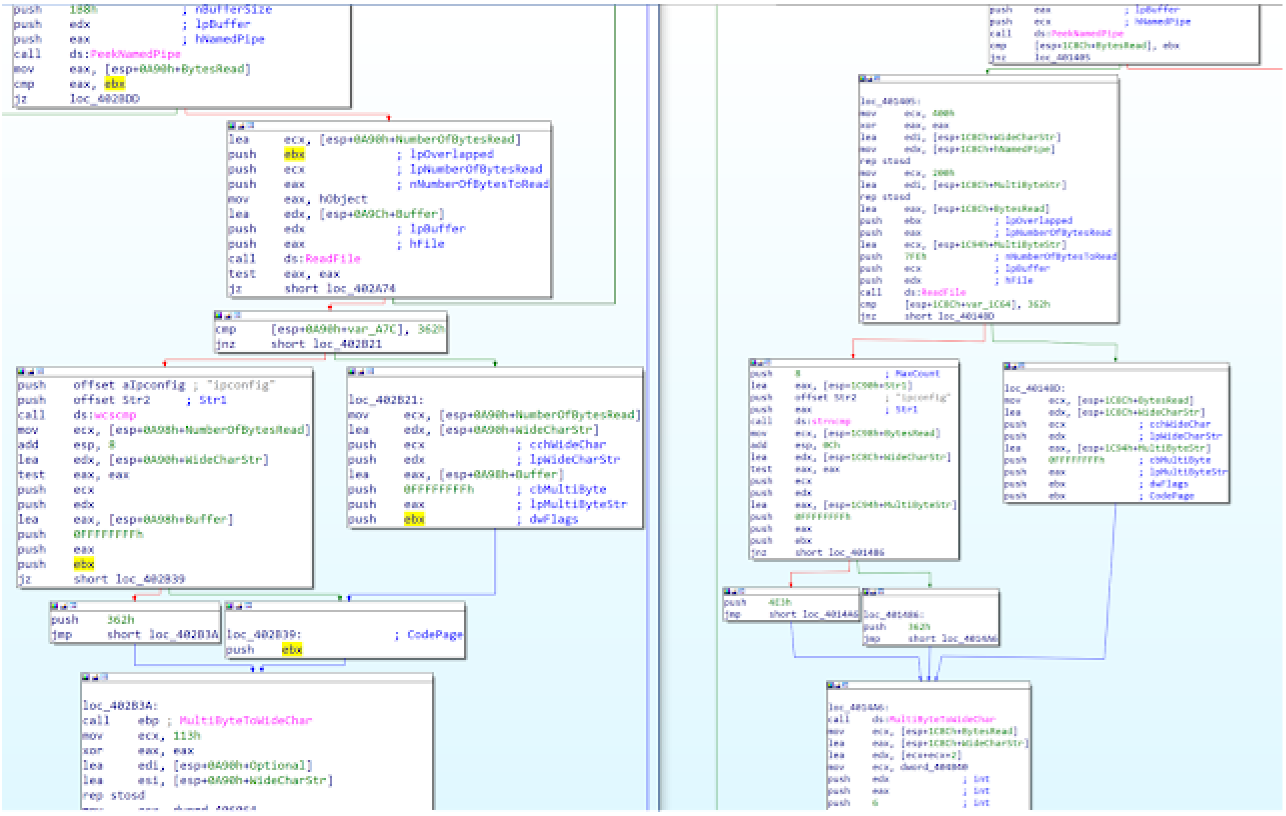
The developer rewrote a large part of the code however the workflow is the same as previously and some features are copy/paste. The binary is compiled with the MFC framework.

The biggest change is the network communication with the C2 server. The malware does not use a raw socket anymore but all the communications are performed with WinInet. The malware performs connection to the C2 server by using InternetOpenA() with an hardcoded User-Agent: "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322". Note the missing parenthesis at the end of the User-Agent. This typo will be there till today.



This variant has exactly the same features as the previous variant: file listing, OS version getting, process killing, drive listing, execution via ShellExecuteW(), execution via named pipe, cleaning, file removal, file downloading.

Here is an example of code similarities on the execution via named pipe function. On the left a sample from Bisonal 2014 and on the right Bisonal 2011. The code is not exactly the same but the workflow and some constants are similar.



Hard-coded Identifiers & URL pattern

In this new version, we identify three hard-coded identifiers:

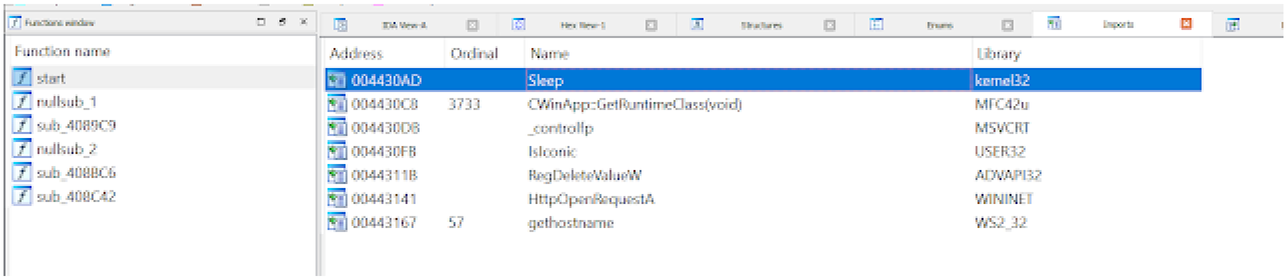
- Campaign ID: an ID put in the exfiltrated data with the hostname and the OS version. We assume this ID is used to identify the campaign and the target by the operator;
- Malware ID: used to generate the first "word" of the URL. We assume this ID is used to identify the malware version (from a network protocol point of view);
- Third ID: used to generate the end "word" of the URL. It generally looks like a file name.

The URL pattern is the following: hxxp://C2_domain:PORT/MalwareIDVictimIP**ThirdID**

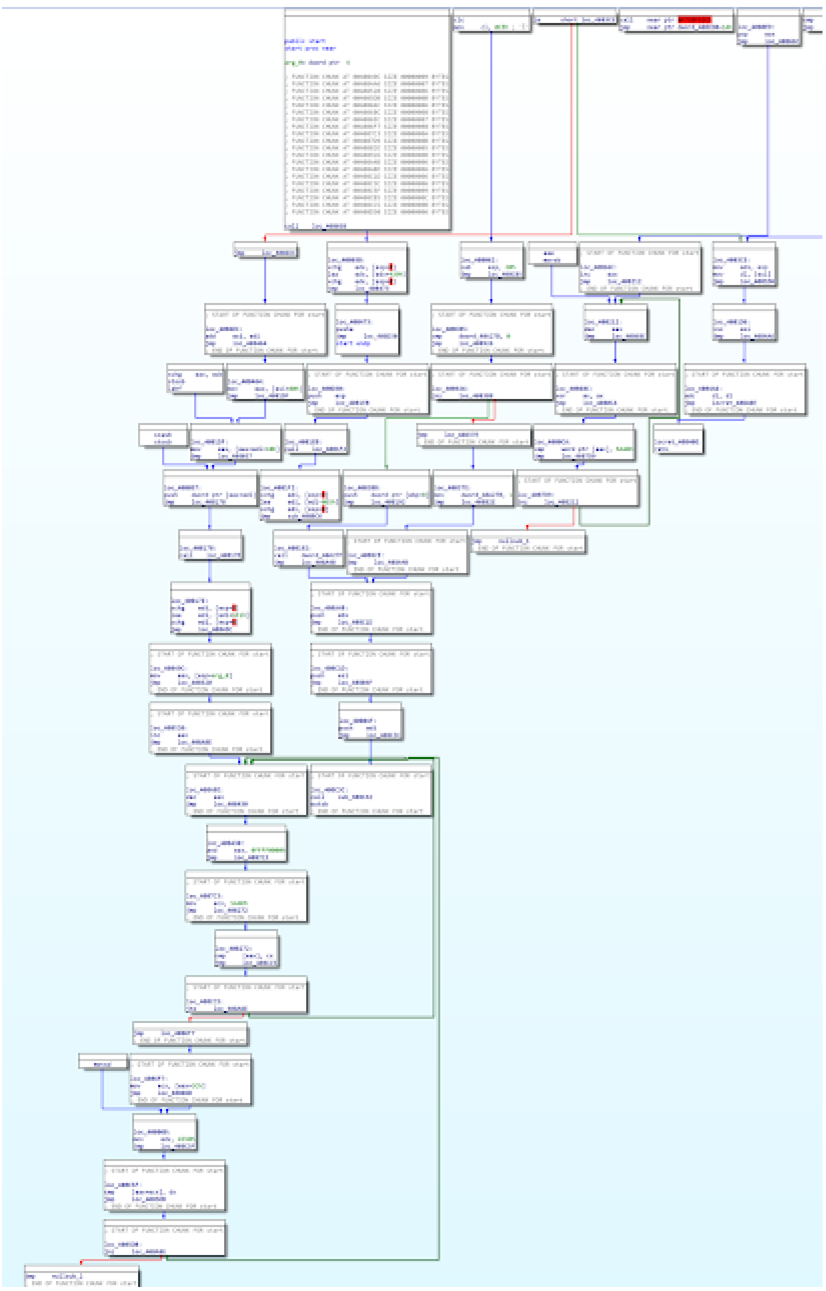
SHA256: c6baef8fe63e673f1bd509a0f695c3b5b02ff7cfe897900e7167ebab66f304ca
C2 URL: hxxp://www[.]hosting[.]tempors[.]com:443/av9d0.0.0.0akspbv.txt

2016: More packers

In 2016, the developer implemented a new way of packing Bisonal. An initial static analysis immediately shows an executable with very little information. IDA Pro only shows five functions and almost no imports.



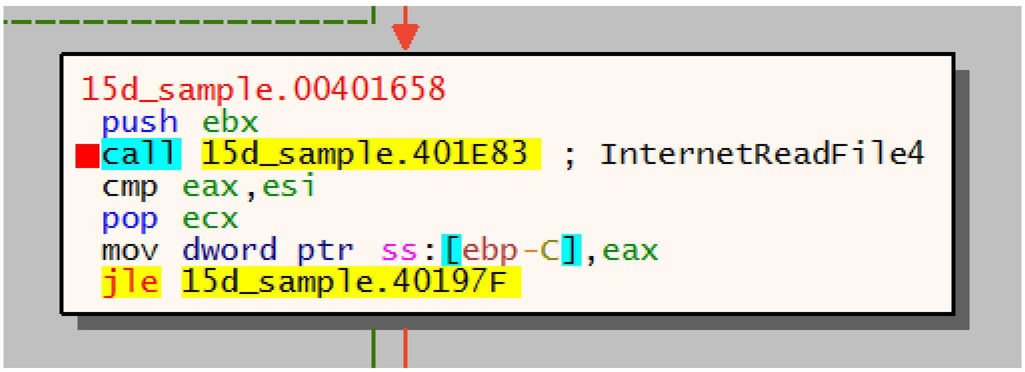
Looking at the few functions available it becomes clear the packer uses several anti-analysis tricks. In the unpacking stage, the malware has a lot of useless jumps and calls which makes the code tracking in the debugger harder.



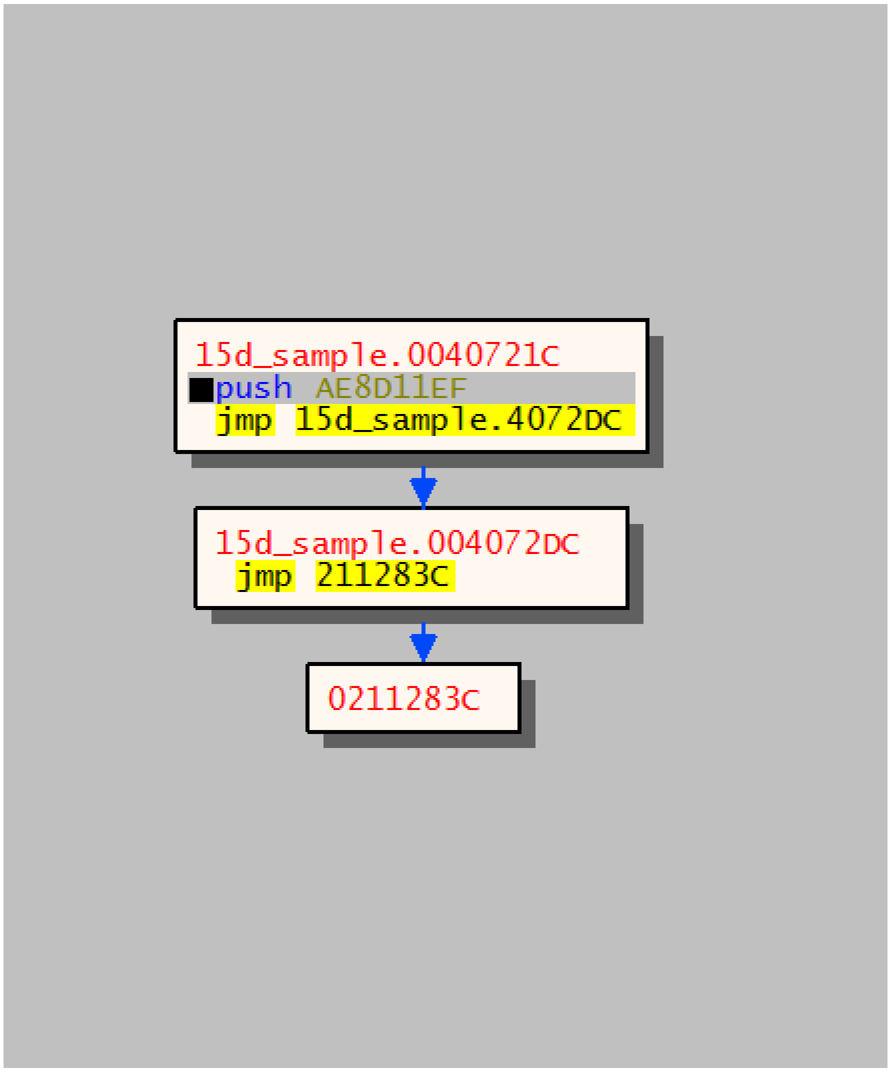
After the unpacking is done the malware continued to use several anti-analysis measures. There are almost no direct calls to functions. It is common during the unpacking process to find useless code, like sequences of one instruction followed by a jump or increments in register values almost immediately followed by decrements. The initial unpacking is based on the manipulation of the return addresses pushed in the stack and the ordering of the data within the .text section. A second stage will allocate memory and unpack code into it, which finally will unpack code into a section that is originally empty called .textbss. This is where the core of the malware will be.

All API calls are made through a dispatcher function. Which is not called directly either, before this function is called it goes through a series of jumps and the stack is filled with encoded offset values.

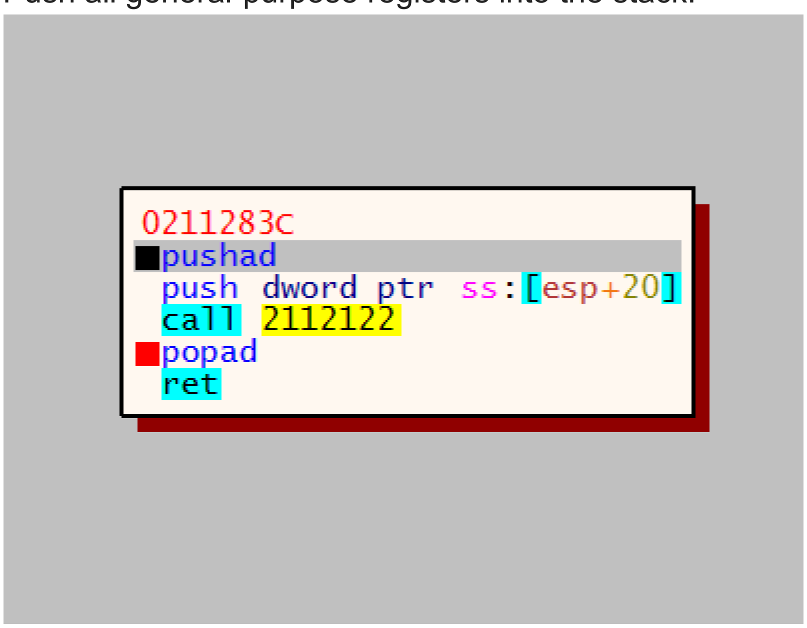
The call of the jump table entry:



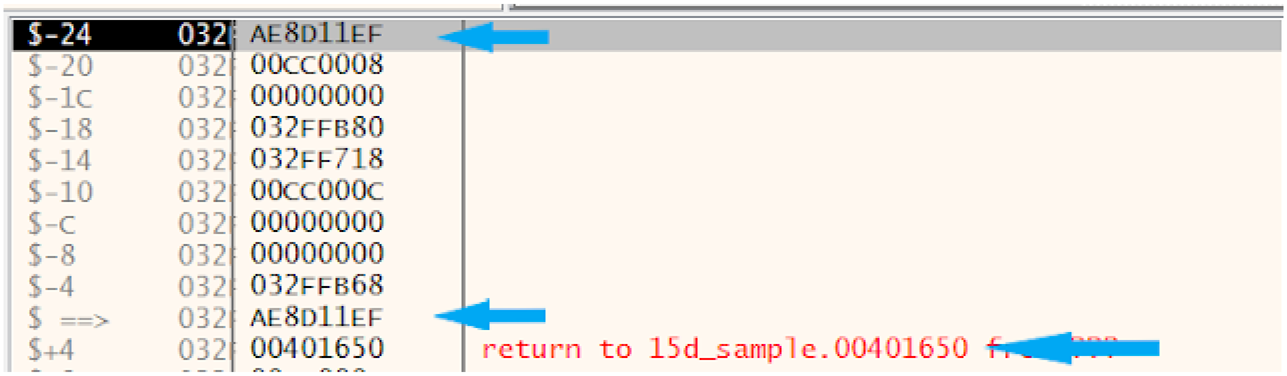
Push parameter for dispatch function into the stack:



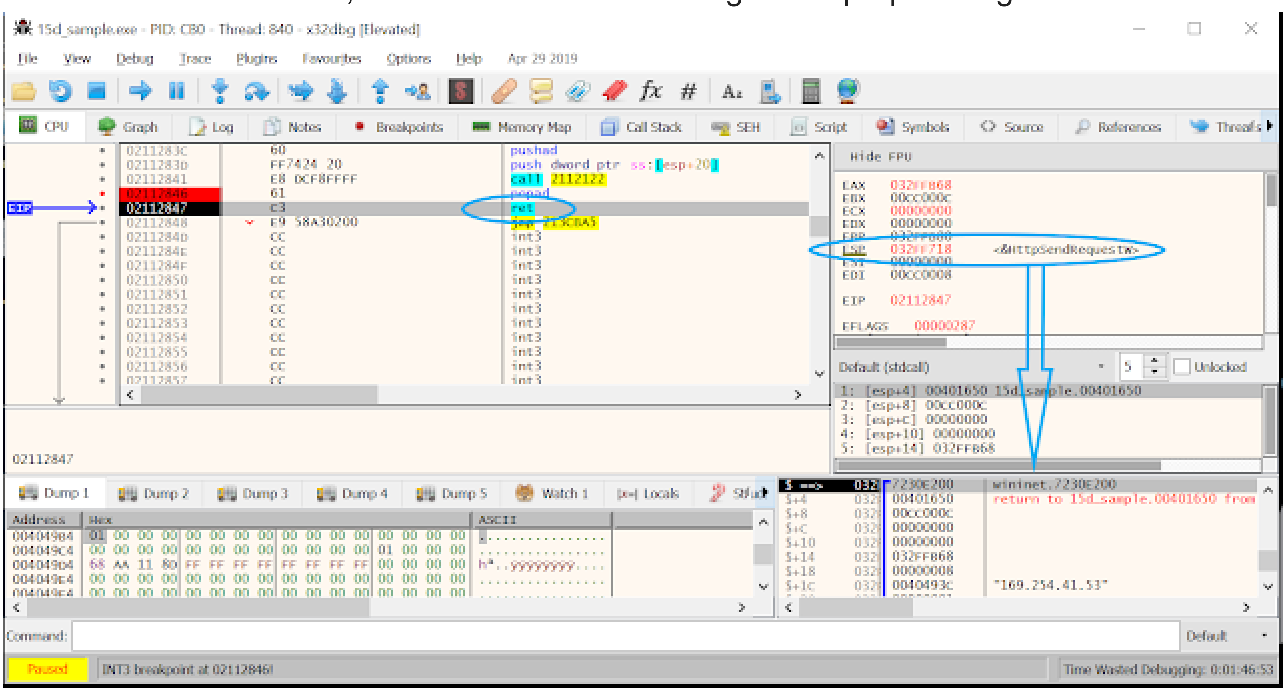
Push all general-purpose registers into the stack:



Before calling the actual dispatch function, all registers are saved to the stack, by doing this the offset value is no longer on the top of the stack so the malware needs to put it back on the top of the stack.

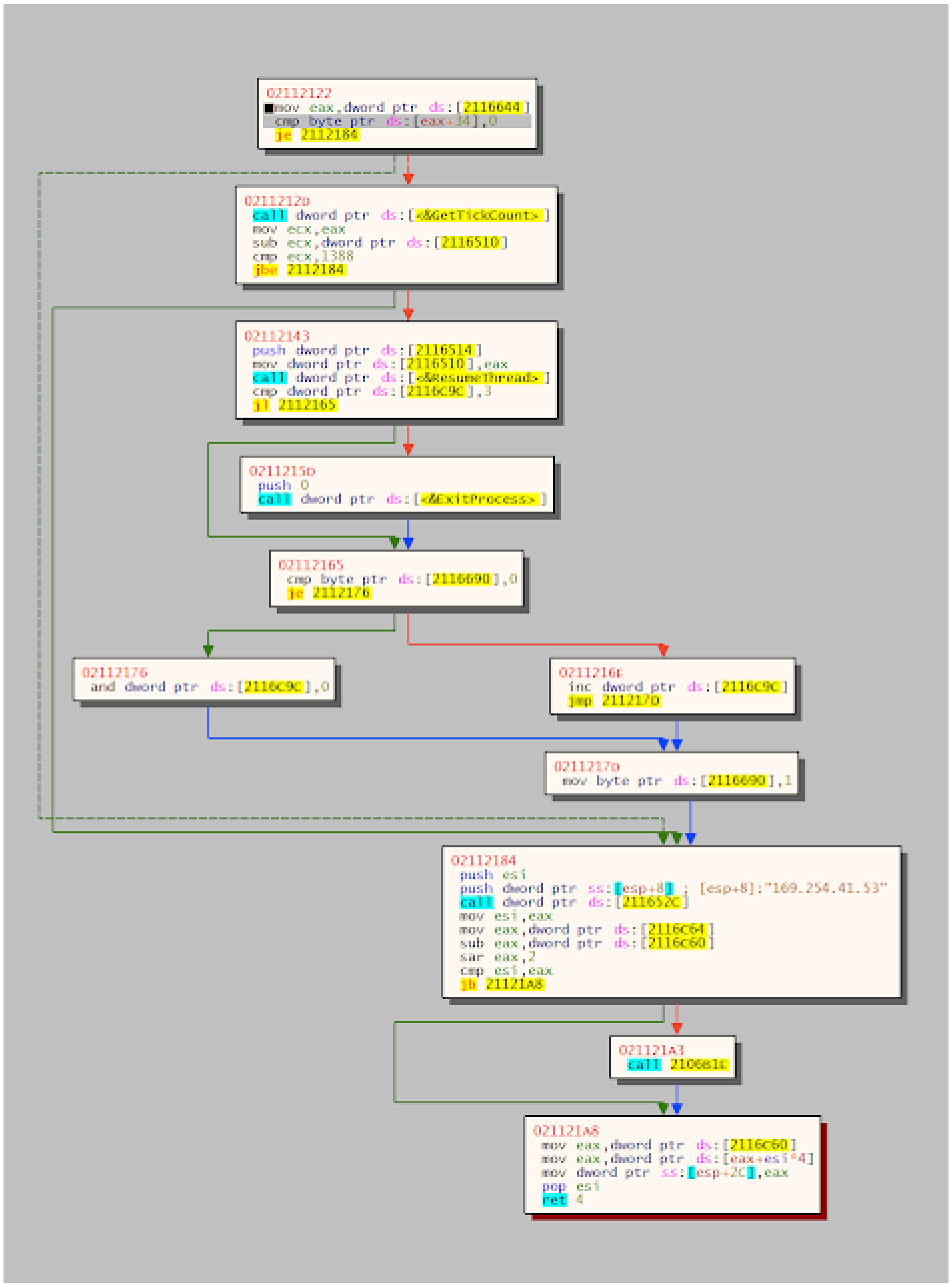


At this time and just before the argument in the stack we also have the return address, inside the core of the malware. The dispatcher function will push the desired API function address into the stack. Afterward, it will do the same for the general-purpose registers.



After calling the dispatcher function the malware will first restore the generic purpose registers from the stack, thus leaving the API function address at the top of the stack. Logically, after the ret instruction is executed the code will jump into the API function.

This mechanism allows the malware to execute API functions without ever using the Call instruction, making it difficult to perform the analysis. The other side effect is that even after the code is unpacked if the analyst tries to dump it and analyze it statically, it will be hard for the disassembler to understand the code.



The dispatcher function has other tricks up its sleeve. Every time it is called it will use the anti-debug GetTickCount() to check if it is being debugged. If there is a discrepancy in the timing it will terminate the process. The termination can be as simple as a call to ExitProcess(), or it will first resume a thread that will display a message to the user. So that it ensures the thread has a chance to run, it will return the API call sleep() no matter what was originally requested. Once sleep() is executed, the error message thread will have a chance to be executed and will terminate the process.

From the functionality point of view, there aren't many differences between the 2014 versions. Always using three hard-coded identifiers mentioned previously but with different values.

SHA256: 15d5c84db1fc7e13c03ff1c103f652fbced5d1831c4d98aad8694c08817044cc
C2 URL: hxxp://emsit[.]serveirc[.]com/ks8d0.0.0.0akspbu.txt

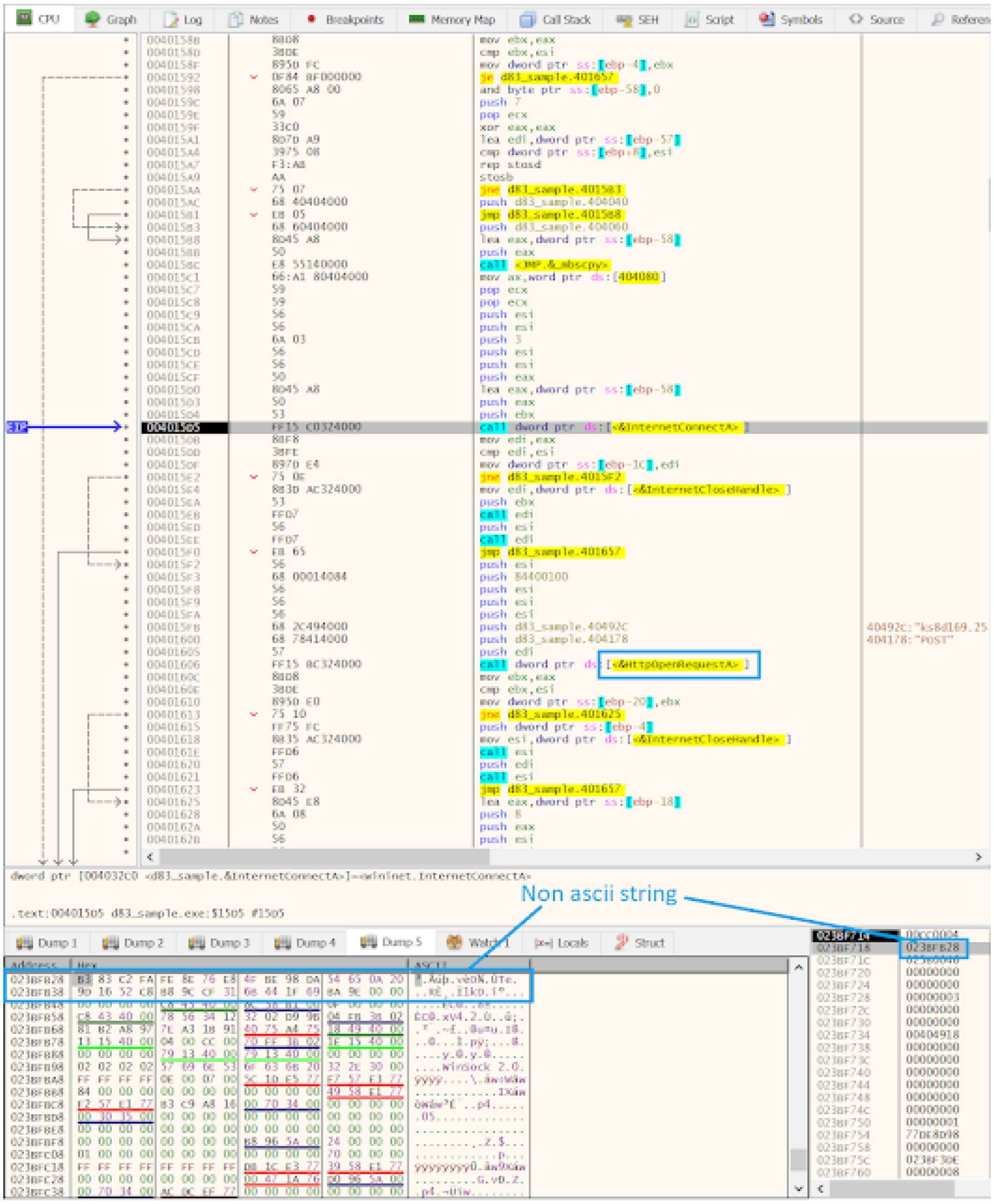
2018: I miss you

During 2018, the attackers used a mix of samples using the MFC framework or the Visual C libraries. The registry key used for the persistence is now named "mismyou".

In September 2018, the developer made a mistake. Normally on this variant of Bisonal the domain names are encoded. However, the developer forgot to obfuscate the strings and put them in clear text into the variables but the deobfuscation function is still executed:

```
loc_40140A:                ; hKey
push    [ebp+phkResult]
call    ds:RegCloseKey
push    1Eh
push    offset a21kmgMyHomeipN ; "21kmg.my-homeip.net"
lea     eax, [ebp+var_8]
push    4
push    eax
mov     [ebp+var_8], 12345678h
call    DeObfuscation
push    1Eh
push    offset Source   ; "21kmg.my-homeip.net"
lea     eax, [ebp+var_8]
push    4
push    eax
call    DeObfuscation
sub     nServerPort, 0Ah
push    ebx             ; Time
call    ds:time
```

The mistake has for effect to destroy the domain and generate garbage strings. The malware will try to perfect connection to this bad domain (hxxp://硟满v鐿緲赦e ?r雀溝1kdi 癹:70/ks8d0.0.0.0akspbu.txt). You can see here a screenshot of the debugger trying to perform a connect on it:



SHA256: 92be1bc11d7403a5e9ad029ef48de36bcff9c6a069eb44b88b12f1efc773c504
C2: kted56erhg[.]dynssl[.]com

SHA256: d83fbe8a15d318b64b4e7713a32912f8cbc7efbfae84449916a0cbc5682a7516
C2 fail: hxxp://硟满v鐿緲赦e ?r雀溝1kdi簦:70/ks8d0.0.0.0akspbu.txt

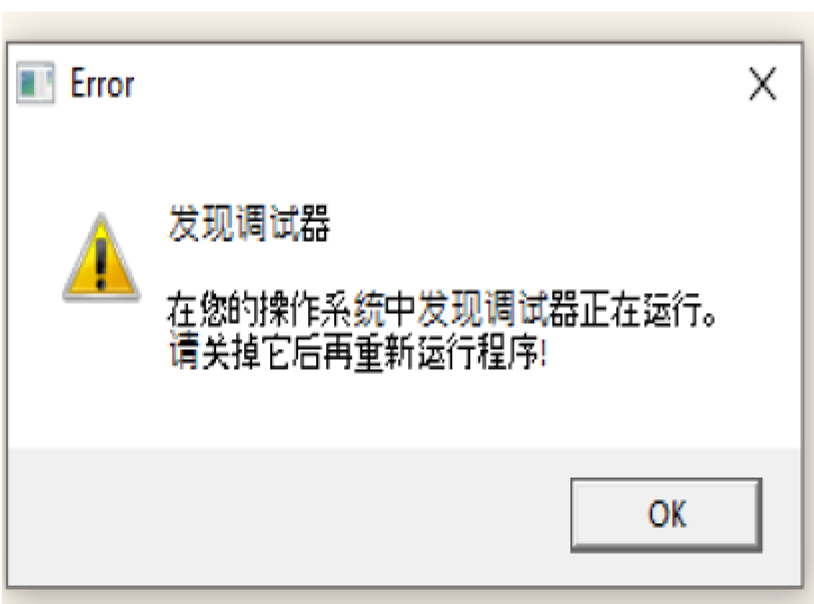
2019 - Office Extension and a new packer

Packer

Static analysis of this executable shows only two functions, but a regular number of imports.

This time the packer shares some of the characteristics from the advanced one used in 2016.

There is a lot of useless code, including jumps and bswap operations. Upon detecting a debugger attached to it, the malware will display the message below and terminate the execution.



This message translates to "The debugger was found to be running in your operating system. This turns it off before running the program again!".

This packer also hides the calls to API functions. This time instead of using a dispatcher function, the malware pushes the arguments into the stack as usual but will then perform a call to a jump table built during the unpacking, in the .text section memory region.

```
0045B05F  90                 nop
0045B060  72 00              jb d7_sample.45B062
0045B062  E9 E91AF571        jmp <wininet.HttpOpenRequestA>
0045B067  7E A9              jle d7_sample.45B012
0045B069  D6                 salc
0045B06A  D4 9A              aam 9A
0045B06C  1B22               sbb esp,dword ptr ds:[edx]
0045B06E  C2 C497            ret 97C4
0045B071  304F 27            xor byte ptr ds:[edi+27],cl
0045B074  CE                 into
0045B075  66:45              inc bp
0045B077  56                 push esi
0045B078  5E                 pop esi
0045B079  74 00              je d7_sample.45B07B
0045B07B  E9 100AE971        jmp <wininet.InternetOpenA>
0045B080  A5                 movsd
0045B081  E2 B0              loop d7_sample.45B033
0045B083  24 99              and al,99
0045B085  90                 nop
0045B086  5B                 pop ebx
0045B087  35 471FDEBD        xor eax,BDDE1F47
0045B08C  81DB 4684C4EE      sbb ebx,EEC48446
0045B092  71 00              jno d7_sample.45B094
0045B094  71 00              jno d7_sample.45B096
0045B096  E9 153BE671        jmp <wininet.InternetReadFile>
0045B09B  E7 F9              out F9,eax
0045B09D  74 21              je d7_sample.45B0C0
0045B09F  40                 inc eax
0045B0A0  DC2A               fsubr st(0),qword ptr ds:[edx]
0045B0A2  80AE B34C1758 B3   sub byte ptr ds:[esi+58174CB3],B3
0045B0A9  CB                 ret far
0045B0AA  D95A 9D            fstp dword ptr ds:[edx-63],st(0)
0045B0AD  59                 pop ecx
0045B0AE  73 00              jae d7_sample.45B0B0
0045B0B0  87D3               xchg ebx,edx
0045B0B2  87DA               xchg edx,ebx
0045B0B4  E9 07C8EA71        jmp <wininet.InternetConnectA>
0045B0B9  A2 5AC6C5DB        mov byte ptr ds:[DBC5C65A],al
0045B0BE  F655 68            not byte ptr ss:[ebp+68]
0045B0C1  1E                 push ds
0045B0C2  BE 7E5A0114        mov esi,14015A7E
0045B0C7  DEEB               fsubp st(3),st(0)
0045B0C9  0E                 push cs
0045B0CA  024F 4E            add cl,byte ptr ds:[edi+4E]
0045B0CD  74 00              je d7_sample.45B0CF
0045B0CF  90                 nop
0045B0D0  E9 DB5E5E77        jmp <user32.LoadIconW>
```

Even though a call is made, these are not functions, in fact, most of the code in this jump table is useless except for the last instruction of each entry. Each entry finishes with a jmp instruction into the respective API function. Effectively the malware doesn't do any call to API functions, it always performs a jump. The return address is loaded into the stack when the malware does a call to the jump table. The end result is the same has in the packer from 2016, but with a simpler mechanism.

The majority of the code was moved into a packed area. The malware configuration (such as C2 server and the User-Agent) is outside that area. The packer uses a thread-local storage (TLS) callback to unpack some of the code. At this stage, it uses in-place unpacking avoiding memory allocations. One of the anti-analysis features included in this packer is the lack of calls to API functions. In the early stages of execution, the malware loads the libraries and retrieves the addresses from functions it needs.

Feature-wise, there is no change when compared with the 2016 version, in fact when compared the C2 beaconing functions even share some of the offsets.

Office Extension

In 2019, the actor behind Bisonal used a new way to deploy the machine on the target's systems. They sent a malicious RTF document to the targets with an exploit targeting the CVE-2018-0798 (Microsoft's Equation Editor vulnerability). The purpose of the shellcode was not to execute the malware (as it is usual) but simply to drop it in the %APPDATA%\microsoft\word\startup\ repository with the .wll extension.

The libraries in this directory with this specific extension will be loaded as a Microsoft Office extension. So next time the user opens an Office application, the malware will be loaded and executed. The purpose of the malware is to deploy Bisonal on the infected system (tmp\tmplogon.exe) and to create a Run registry key in order to execute Bisonal at the next reboot of the system.

We think the purpose of this multistage execution is an anti sandbox technique. If you look at the report after executing the malicious document, you only see one action: the .wll file creation. The user also needs to open an Office application and finally a reboot is needed in order to execute the real payload: Bisonal.

Bigger is better

We identified a version of Bisonal using Office extension with a really specific behavior during the installation of the malicious payload. The dropper appends 80MB of binary data at the end of the Bisonal binary:

```
xor     al, al
pop     esi
mov     esp, ebp
pop     ebp
retn
```

```
loc_10001030:
push    edi
push    0
lea     eax, [ebp+var_4]
push    eax
push    7D000h
push    offset PE_Xored
push    esi
call    WriteFile_
mov     edi, 230000h
lea     ecx, [ecx+0]
```

```
loc_10001050:
push    0
lea     ecx, [ebp+var_4]
push    ecx
push    24h ; '$'
push    offset a00110101001101 ; "00110101001101100100110101001101100 1"
push    esi
call    WriteFile_
dec     edi
jnz     short loc_10001050
```

The binary value is "56MM" is ASCII characters. If we look at the malware, we can see the appended data:

```
0008dda0  30 31 31 30 30 31 30 30  31 31 30 31 30 31 30 30  |0110010011010100|
0008ddb0  31 31 30 31 31 30 30 31  30 30 31 31 30 31 30 31  |1101100100110101|
0008ddc0  30 30 31 31 30 31 31 30  30 31 30 30 31 31 30 31  |0011011001001101|
0008ddd0  30 31 30 30 31 31 30 31  31 30 30 31 30 30 31 31  |0100110110010011|
0008dde0  30 31 30 31 30 30 31 31  30 31 31 30 30 31 30 30  |0101001101100100|
0008ddf0  31 31 30 31 30 31 30 30  31 31 30 31 31 30 30 31  |1101010011011001|
0008de00  30 30 31 31 30 31 30 31  30 30 31 31 30 31 31 30  |0011010100110110|
0008de10  30 31 30 30 31 31 30 31  30 31 30 30 31 31 30 31  |0100110101001101|
0008de20  31 30 30 31 30 30 31 31  30 31 30 31 30 30 31 31  |1001001101010011|
0008de30  30 31 31 30 30 31 30 30  31 31 30 31 30 31 30 30  |0110010011010100|
0008de40  31 31 30 31 31 30 30 31  30 30 31 31 30 31 30 31  |1101100100110101|
0008de50  30 30 31 31 30 31 31 30  30 31 30 30 31 31 30 31  |0011011001001101|
0008de60  30 31 30 30 31 31 30 31  31 30 30 31 30 30 31 31  |0100110110010011|
0008de70  30 31 30 31 30 30 31 31  30 31 31 30 30 31 30 30  |0101001101100100|
0008de80  31 31 30 31 30 31 30 30  31 31 30 31 31 30 30 31  |1101010011011001|
0008de90  30 30 31 31 30 31 30 31  30 30 31 31 30 31 31 30  |0011010100110110|
0008dea0  30 31 30 30 31 31 30 31  30 31 30 30 31 31 30 31  |0100110101001101|
0008deb0  31 30 30 31 30 30 31 31  30 31 30 31 30 30 31 31  |1001001101010011|
```

We are not sure of the purpose of the creation of a huge binary. It could be an anti-analysis technique. Some tools limit the size of the analyzed files. For example, by using the VirusTotal standard API, we cannot upload files bigger than 32MB. We also identified sandboxes that cannot handle big files correctly. Remember, size matters.

Malware code

The developer partially refactored the code. The variant from 2019 keeps exactly the same features. The two main changes are the obfuscation and the network protocol to communicate to the C2 server.

The developers used two different obfuscation algorithms: one for the C2 encoding and one for the data. The C2 encoding is a simple XOR (as in 2012):

```
loc_40136E:
mov     cl, byte ptr aBbbTxtFpgZaozV[eax] ; "bbb;txt{fpg, $;zaoz;vzx"
xor     cl, 15h
mov     name[eax], cl
inc     eax
cmp     eax, edx
jl      short loc_40136E

loc_401382:
xor     eax, eax
test    esi, esi
jle     short loc_40139C

loc_401388:
mov     dl, byte ptr aJjj3yyn3jQnuyK[eax] ; "jjj3yyn3j|qnuy|ktn3~rp"
xor     dl, 1Dh
mov     byte_407184[eax], dl
inc     eax
cmp     eax, esi
jl      short loc_401388
```

The C2 encoding communication is also different. As the data are now sent with the GET method, the data must be in ASCII. That's they add base64 encoding in order to get supported characters in the HTTP query.

For the first time, the developer switched from POST requests to GET requests:

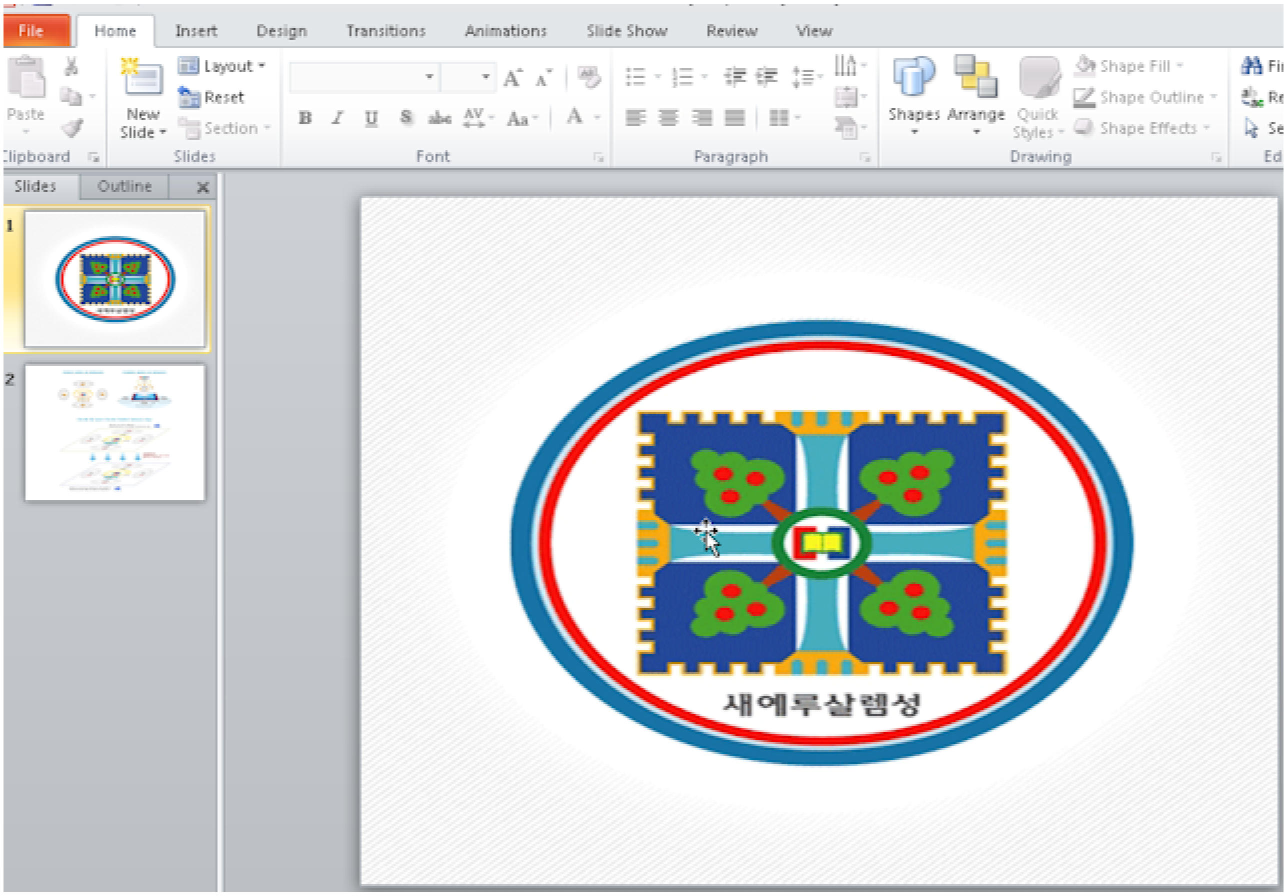
```
add     esp, 0Ch
lea     ecx, [esp+19C14h+szObjectName]
push    0                   ; dwContext
push    84400100h           ; dwFlags
push    0                   ; lplpszAcceptTypes
push    0                   ; lpszReferrer
push    0                   ; lpszVersion
push    ecx                 ; lpszObjectName
push    offset szVerb       ; "GET"
push    ebx                 ; hConnect
call    ds:HttpOpenRequestA
mov     esi, eax
test    esi, esi
jnz     short loc_401940
```

The exfiltrated data is appended to the URL. Here is the pattern:
hxxp://C2_domain/MalwareIDVictimIP**ThirdID**ExfiltratedDataBase64

SHA256:37d1bd82527d50df3246f12b931c69c2b9e978b593a64e89d16bfe0eb54645b0
C2 URL:hxxp://www[.]amanser951[.]otzo[.]com/uiho0.0.0.0edrftg.txt

2020 Business as Usual

Ahnlab, a South Korean software company, simultaneously published a paper regarding Bisonal's activity in South Korea. In this case, the infection vector has changed from previous samples. The initial stage is a binary that drops a decoy document (Powerpoint or Excel document), a VisualBasic script and the packed Bisonal payload. The payload is dropped with a .jpg extension that's been renamed to ".exe." Here is an example decoy document:



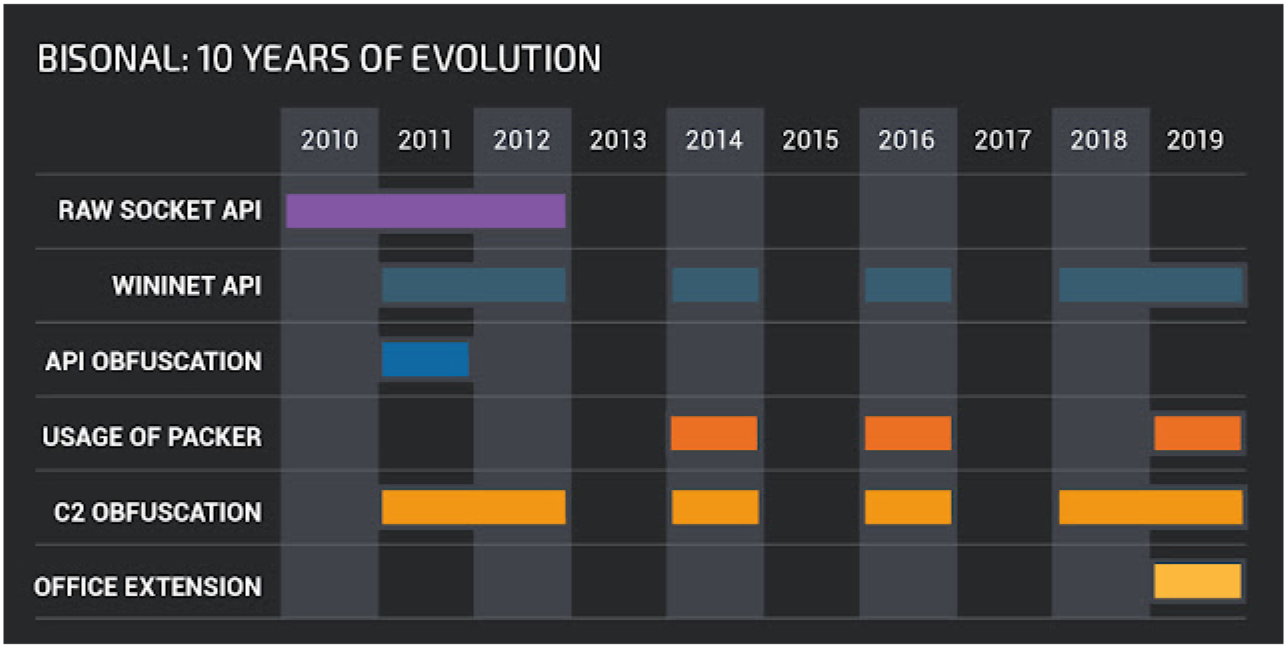
The purpose of the VisualBasic script is to execute the payload. Similar to attacks in 2019, the attacker appends data in order to generate a large binary. Although the malicious part of the binary is only 2MB, the final file is more than 120MB in size, padded out with random data. This may be an attempt to evade antivirus engines that only scan up to a maximum file size. The payload has been packed with a new packer.

The code of Bisonal is similar to the version of 2019. The attacker implements indirect API calls by using GetProcAddress() and LoadLibrary() API.

```
mov     ebp, ds:LoadLibraryW
push    offset LibFileName ; "Shell32.dll"
call    ebp ; LoadLibraryW
mov     esi, ds:GetProcAddress
mov     ebx, eax
push    offset ProcName ; "ShellExecuteW"
push    ebx             ; hModule
call    esi ; GetProcAddress
push    offset aShellexecuteex ; "ShellExecuteExW"
push    ebx             ; hModule
mov     ShellExecuteWAPI, eax
call    esi ; GetProcAddress
push    offset aShchangenotify ; "SHChangeNotify"
push    ebx             ; hModule
mov     ShellExecuteExWAPI, eax
call    esi ; GetProcAddress
push    offset aShlwapiDll ; "Shlwapi.dll"
mov     SHChangeNotifyAPI, eax
call    ebp ; LoadLibraryW
push    offset aShdeletevaluea ; "SHDeleteValueA"
push    eax             ; hModule
call    esi ; GetProcAddress
push    offset aWininetDll ; "wininet.dll"
mov     SHDeleteValueAAPI, eax
call    ebp ; LoadLibraryW
mov     ebx, eax
push    offset aInternetopena ; "InternetOpenA"
push    ebx             ; hModule
call    esi ; GetProcAddress
push    offset aInternetconnec ; "InternetConnectA"
push    ebx             ; hModule
mov     InternetOpenAAPI, eax
call    esi ; GetProcAddress
push    offset aInternetcloseh ; "InternetCloseHandle"
push    ebx             ; hModule
mov     InternetConnectAPI, eax
```

Sha256: b7ef3ec4d9b0fd29c86c9a4b2a94819a80c83e44cdc47a9091786d839be6a7c4
C2: imbc[.]onthewifi[.]com

Bisonal timelines summary



Conclusion

The actor behind Bisonal is clearly motivated and has an interest in Russian, Korean and Japanese victims. The development of Bisonal has been active for more than a decade. We have observed the code evolving with the different publications but also with the evolution of Microsoft Windows.

However, specific functions are still used today, many years after the original implementation of the Bional malware. Even if Bisonal could be considered as simple with less than 30 functions, it has spent its life targeting sensitive entities in both the public and private sectors. Some campaigns were even mentioned on mainstream media against military entities within the mentioned regions.

During the decade of activities, we also can see mistakes and rollbacks from the attackers. For example, in one campaign they put the domain name of the C2 server in plaintext in the malware which had the function to generate a non-ASCII string for the C2 servers once decoded. In this condition, the malware cannot work on the compromised system. Even after so many years of activities, the attackers make mistakes.

We don't see any reason why this actor will stop in the near future. With this investigation and the analysis of this decade of activity, we hope to force this actor to innovate by providing a better understanding of his arsenal and more specifically how Bisonal works.

Coverage

Ways our customers can detect and block this threat are listed below.

Product	Protection
AMP	✓
Cloudlock	N/A
CWS	✓
Email Security	✓
Network Security	✓
Stealthwatch	N/A
Stealthwatch Cloud	N/A
Threat Grid	✓
Umbrella	✓
WSA	✓

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors. Exploit Prevention present within AMP is designed to protect customers from unknown attacks such as this automatically.

Cisco Cloud Web Security (CWS) or Web Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

Network Security appliances such as Next-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS), Cisco ISR, and Meraki MX can detect malicious activity associated with this threat.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

IOCs

OSQuery

Cisco AMP users can use Orbital Advanced Search to run complex OSqueries to see if their endpoints are infected with this specific threat. For specific OSqueries on this threat, click below:

- Bisonal File Path Detected
- Bisonal Registry Detected

SHA256:

0cf9d9e01184d22d54a3f9b6ef6c290105eaa32c7063355ca477d94b130976af
7dc58ff4389301a6eccc37098682742b96e5171d908acdeb62aeaa787496c80a
0ff88a6cd7dcd27f14ebb7b2c97727b81e1aa701280d1164685c52c234e4a9df
8252f2cdedf16f404d43c81d005ea8ebb10594477f738e40efacf9013e1470d2
915ad316cfd48755a9e429dd5aacbee266aca9c454e9cf9507c81b30cc4222e5
1128d10347dd602ecd3228faa389add11415bf6936e2328101311264547afa75
92be1bc11d7403a5e9ad029ef48de36bcff9c6a069eb44b88b12f1efc773c504
15d5c84db1fc7e13c03ff1c103f652fbced5d1831c4d98aad8694c08817044cc
9638e7bb963ac881bd81071d305dea91b040536c55b7ee79b526b8afcfad6972
1e66579b856cd331518d67c351bcb2b102399d8ade53370797228b289e905dc1
979d4e6665ddd4c515f916ad9e9efd9eca7550290507848c52cf824dfbd72a7e
22b3a86f91d2eb5a8a1e1cdc044bcf6aca898663071be5233bac00c0f0d3c001
9c86c2dd001c47b933c6b5f43c8f87a6d0c01c066e3520e651fab51d19355d3c
2c1e0facf563bb2054d9a883144ef9bad77ba75cdb46cc80843821c363c0a9dc
a4a5c60a392d236b76907f58597e83ba9c9d4cfc6a4502ef3e0e149b8710a0c6
359835c4a9dbe2d95e483464659744409e877cb6f5d791daa33fd601a01376fc
b1da7e1963dc09c325ba3ea2442a54afea02929ec26477a1b120ae44368082f8
37d1bd82527d50df3246f12b931c69c2b9e978b593a64e89d16bfe0eb54645b0
b75c986cf63e0b5c201da228675da4eff53c701746853dfba6747bd287bdbb1d
43459f5117bee7b49f2cee7ce934471e01fb2aa2856f230943460e14e19183a6
b85e4168972b28758984f919aef2ce0fde271ee1f0863510e521a2920fcc658e
43606116e03672d5c2bca7d072caa573d3fc2463795427d6f5abfa25403bd280
bd1a9b148580dad430683639b747d1c49932db5d8f6eb2d90e2583af976810dc
436fc9530015c2d2b952a16d2a3dfa202d1cb1c577b580811b9b48355855591b
c5496dc3fa96b657ab4467c551877bbced56fd07c00c7ccb199c1794235bf710
444e864a3bb2abb1edccab4a5cd45bc0039f2a48e01615b2719da65a40a5140e
c6baef8fe63e673f1bd509a0f695c3b5b02ff7cfe897900e7167ebab66f304ca
cdba1a69d75f3e2256dccc16255aef07ded41c257b2cc95ccb801a0063445926
5caada5737b0a6c8c8f8a27bfcd0fb2221af68a4856278c3919b37279daa7409
d19b85891dd0f83808b70fbe68a56a64e828611dfe53d04a6c1c211f1352b5b5
6676934d7f214cb256407400357c1f7ead69a523b3017f6a5bc30d06a11a8305
d7692a71b85c869ee11647b80ea6d42b2e4303233c525a8fa7e6bec3599e2c8b
67e286c7308dda5cd8fe4a1340f354927e5791ce6ef0ef02c93a4e063e11c4ad
d83fbe8a15d318b64b4e7713a32912f8cbc7efbfae84449916a0cbc5682a7516
6c714653a8fa54eef1de2f0148e5e8cf514907f6f523bf09c8ee126bebcdbdcc

dd88b31275b7079899d945fc6de2dceaf7e8fc143ef24be5bb336585ddf6af1e
6cc4707942f9323347c95066a43b30f874f1b1c783960cf8ed9ecf5914f85ba7
eb7681c653ef1942103cd3272fd124eaf73e79bb830be978535c18b73c87b985
6ef4df8460ba57b836f52a9a73e2d739a3f2aa832bec6b663af53b55dc74a63d
effd31b11bdc6486082967c2d8e53d979e59a88ba28e68a1c94f5a064a8a966d
6f4a1b423c3936969717b1cfb25437ae8d779c095f158e3fded94aba6b6171ad
6f8bbea18965b21dc8b9163a5d5205e2c5e84d6a4f8629b06abe73b11a809cca
f3a30e5f8bfd0f936597bcef7cb43df11ec566467001dff9365771900e90acb1
77a36530555eada268238050996839bd34670e8bfda477c30d9dd66574625f59
f9302b7ecc32b891edeaf61353dc5e976832b7104ec0d36f1641f1f40cf6fe12
799d858ff77c29684fc1522804ed45c24171484d9618211c817df01424bc981a
23d263b6f55ac81f64c3c3cf628dd169d745e0f2b264581305f2f46efc879587
72f6a54d0d09a16e6fde9800aa845cd1866001538afb2c8f61f3606f5e13f35a
4bad5898373eb644662a8c1d5d5c674e2558908e34bb2fd915f3350b0f28752b
b7ef3ec4d9b0fd29c86c9a4b2a94819a80c83e44cdc47a9091786d839be6a7c4

C2 servers:

0906[.]toh[.]info
dnsdns1[.]PassAs[.]us
euiro8966[.]organiccrap[.]com
jennifer998[.]lookin[.]at
kfsinfo[.]ByInter[.]net
kted56erhg[.]dynssl[.]com
mycount[.]MrsLove[.]com
since[.]qpoe[.]com
usababa[.]myfw[.]us
v3net[.]rr[.]nu
www[.]amanser951[.]otzo[.]com
www[.]amanser951.otzo[.]com
137[.]170[.]185[.]211
196[.]44[.]49[.]154
21kmg[.]my-homeip[.]net
61[.]90[.]202[.]197
61[.]90[.]202[.]198
69[.]197[.]149[.]98
agent[.]my-homeip[.]net
applejp[.]myfw[.]us
dnsdns1[.]PassAs[.]us
emsit[.]serveirc[.]com
etude[.]servemp3[.]com
euiro8966[.]organiccrap[.]com

faceto[.]UglyAs[.]com
games[.]my-homeip[.]com
hansun[.]serveblog[.]net
hxxp://硟满v鐿緲赦e ?r雀溝1kdi焱:70/ks8d0.0.0.0akspbu.txt
indbaba[.]myfw[.]us
kazama[.]myfw[.]us
kreng[.]bounceme[.]net
kted56erhg[.]dynssl[.]com
mycount[.]MrsLove[.]com
navego[.]serveblog[.]net
shinkhek[.]myfw[.]us
wew[.]mymom[.]info
www[.]hosting[.]tempors[.]com
www[.]nayana[.]adultdns[.]net
www[.]dds.walshdavis[.]com
imbc[.]onthewifi[.]com