Naikon APT: Cyber Espionage Reloaded

research.checkpoint.com/2020/naikon-apt-cyber-espionage-reloaded

May 7, 2020

May 7, 2020

Introduction

Recently Check Point Research discovered new evidence of an ongoing cyber espionage operation against several national government entities in the Asia Pacific (APAC) region. This operation, which we were able to attribute to the **Naikon APT group**, used a new backdoor named **Aria-body**, in order to take control of the victims' networks.

In 2015, an extensive report by ThreatConnect and Defense Group revealed the APT group's infrastructure and even exposed one of the group's members. Since this report, no new evidence has come to light of further activity by the group, suggesting that they had either gone silent, increased their emphasis on stealth, or drastically changed their methodology of operations. That is, until now.

In the following report, we will describe the tactics, techniques, procedures and infrastructure that have been used by the Naikon APT group over the 5 years since the last report, and offer some insight into how they were able to remain under the radar.

Targeting

By comparing with previously reported activity, we can conclude that the Naikon APT group has been persistently targeting the same region in the last decade. In operations following the original 2015 report, we have observed the use of a backdoor named **Aria-body** against several national governments, including **Australia, Indonesia, the Philippines, Vietnam, Thailand, Myanmar** and **Brunei**.

The targeted government entities include ministries of foreign affairs, science and technology ministries, as well as government-owned companies. Interestingly, the group has been observed expanding its footholds on the various governments within APAC by launching attacks from one government entity that has already been breached, to try and infect another. In one case, a foreign embassy unknowingly sent malware-infected documents to the government of its host country, showing how the hackers are exploiting trusted, known contacts and using those them to infiltrate new organizations and extend their espionage network.

Given the characteristics of the victims and capabilities presented by the group, it is evident that the group's purpose is to gather intelligence and spy on the countries whose Governments it has targeted. This includes not only locating and collecting specific documents from infected computers and networks within government departments, but also extracting data from removable drives, taking screenshots and keylogging, and of course harvesting the stolen data for espionage. And if that wasn't enough, to evade detection when accessing remote servers through sensitive governmental networks, the group compromised and used servers within the infected ministries as command and control servers to collect, relay and route the stolen data.

Targeted countries

Infection Chains

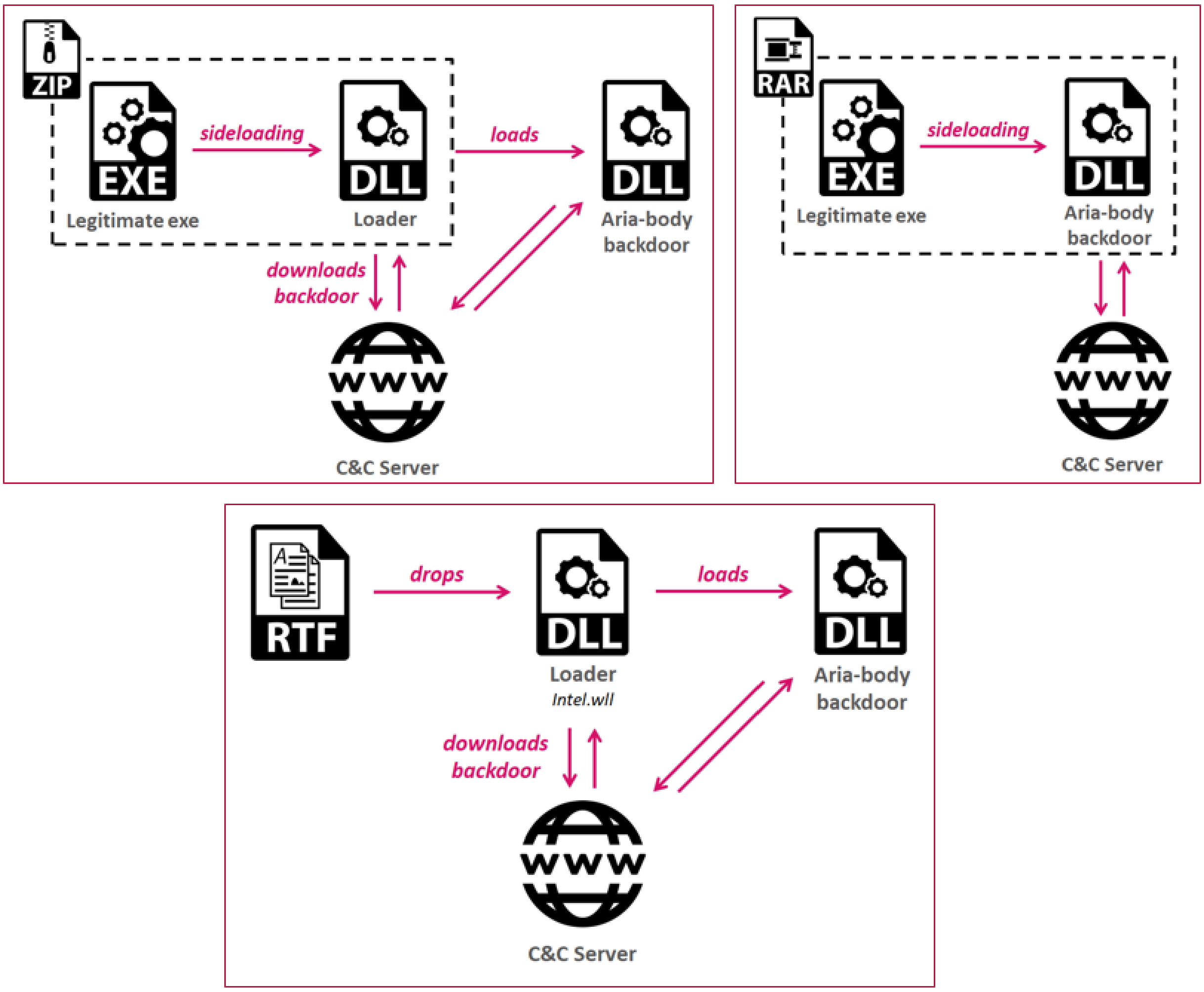
Throughout our research, we witnessed several different infection chains being used to deliver the **Aria-body** backdoor. Our investigation started when we observed a malicious email sent from a government embassy in APAC to an Australian state government, named `The Indians Way.doc` . This RTF file, which was infected (weaponized) with the RoyalRoad exploit builder, drops a loader named `intel.wll` into the target PC's Word startup folder. The loader in turn tries to download and execute the next stage payload from `spool.jtjewifyn[.]com` .

This is not the first time we have encountered this version of the **RoyalRoad** malware which drops a filename named `intel.wll` – the **Vicious Panda** APT group, whose activities we reviewed in March 2020, utilizes a very similar variant.

Overall, during our investigation we observed several different infection methods:

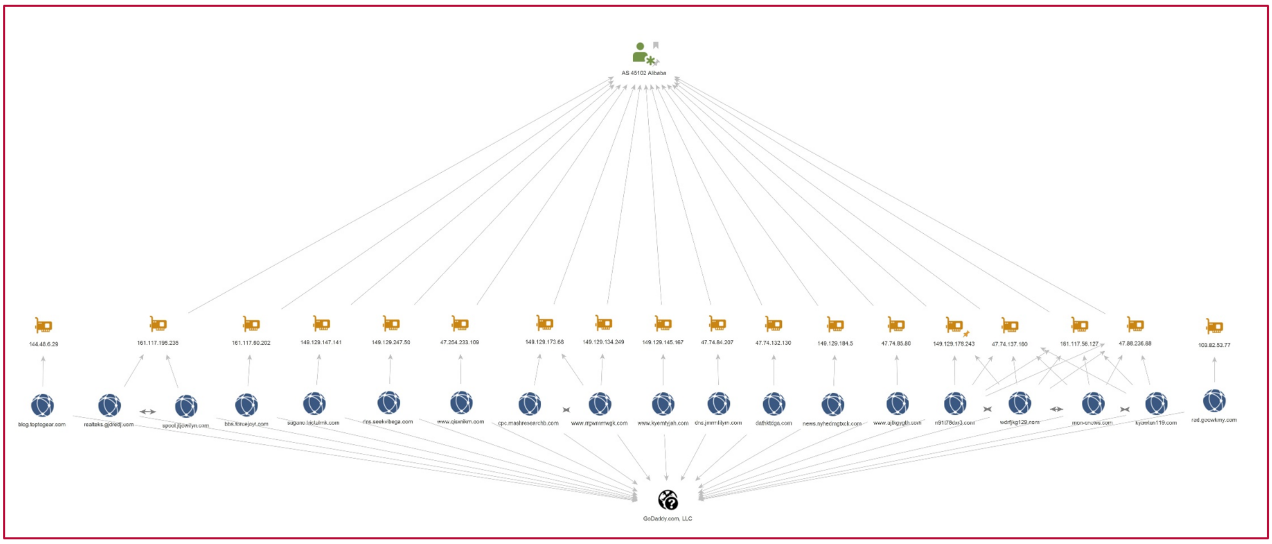
- An RTF file utilizing the **RoyalRoad** weaponizer.
- Archive files that contain a legitimate executable and a malicious DLL, to be used in a DLL hijacking technique, taking advantage of legitimate executables such as **Outlook** and **Avast proxy,** to load a malicious DLL.
- Directly via an executable file, which serves as a loader.

Infection chain examples



Infrastructure

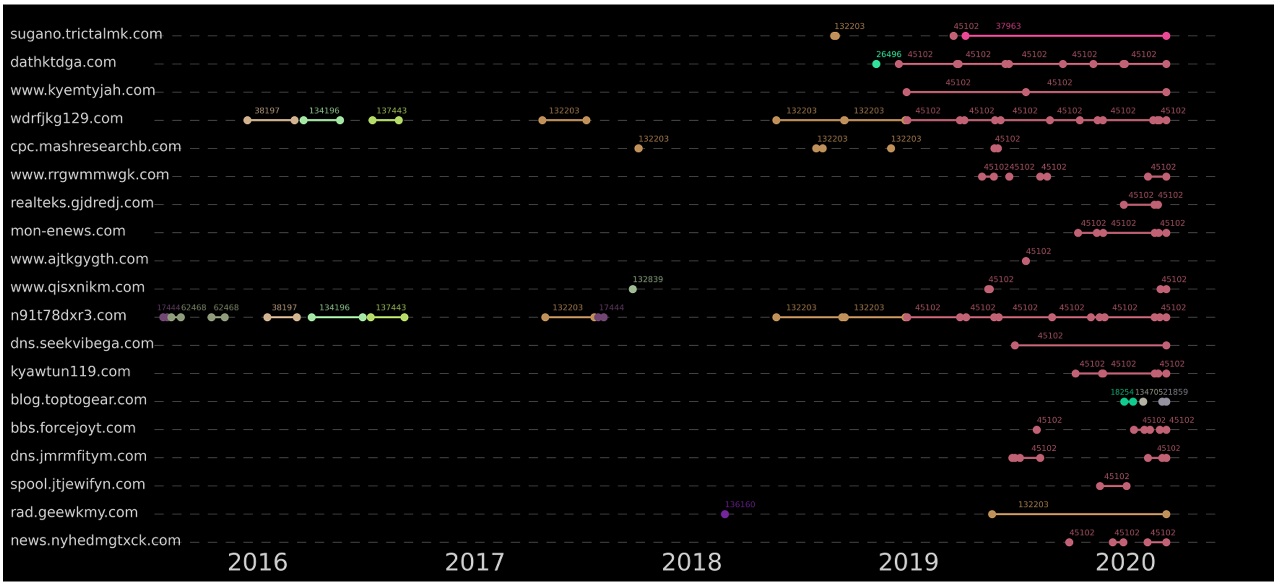
In recent operations, the attackers used the same hosting and DNS services for most of their C&C servers: **GoDaddy** as the registrar and **Alibaba** for hosting the infrastructure. On several occasions, the attackers even reused the same IP address with more than one domain:



Maltego – latest infrastructure overview

A full view of the entire infrastructure is available here.

In order to get a clearer picture of how the attackers operated their infrastructure throughout the years, we have plotted the various malicious domains, according to the ASN they were hosted on, based on periodic passive DNS information. The results are presented in the figure below:



Correlation between domains and ASNs over time

Observations:

- Several domains were utilized for a very long time.
- Multiple domains jumped to the **same** new ASN within a short time frame.
- Since 2019, most of the infrastructure has been concentrated on ASN 45102 (Alibaba).
- In some occasions, the attackers would change the IP address / server, on the same ASN (represented by two consecutive incidental ASN's on the graph).

In addition, one of the more interesting infrastructure properties we observed, is the possible use of hacked government infrastructures as C&C servers. In one of the samples we analyzed, `outllib.dll` (`63d64cd53f6da3fd6c5065b2902a0162`), there is a backup C&C server which is configured as `202.90.141[.]25` – an IP which belongs to the ***Philippines department of science and technology.***

Tool Analysis

In the following section, we will dive into the technical analysis of the **Aria-body** backdoor, utilized throughout the observed activity, as well as an analysis of the loader executable that comes before it.

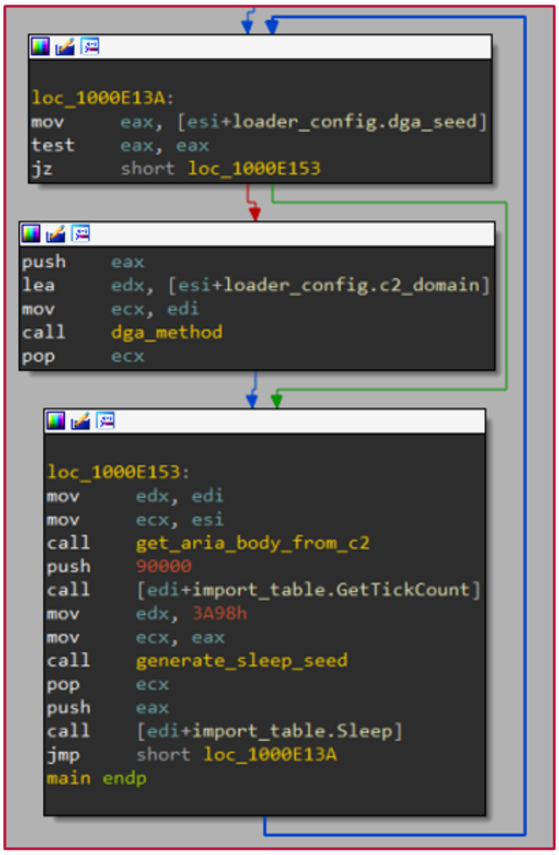
Utilizing the loader at an early stage of an infection allows the attackers to establish a persistent presence on the target's network, as well as perform basic reconnaissance, before using their more advanced tools. While we observed **Aria-body** backdoor variants being compiled as early as 2018, we have observed **Aria-body's** loaders going back to 2017.

Loader Analysis

The functionality of the **Aria-body loader** has not changed significantly since 2017, but the implementation varied from version to version. **This loader appears to be specifically created for the Aria-body backdoor.**

Overall, the loader is responsible for the following tasks:

1. Establish persistence via the `Startup` folder or the `Run` registry key (some variants).
2. Inject itself to another process such as `rundll32.exe` and `dllhost.exe` (some variants).
3. Decrypt two blobs: Import Table and the loader configuration.
4. Utilize a DGA algorithm if required.
5. Contact the embedded / calculated C&C address in order to retrieve the next stage payload.
6. Decrypt the received payload DLL (**Aria-body** backdoor).
7. Load and execute an exported function of the DLL – calculated using `djb2` hashing algorithm.



Main logic of the loader – entering dga_method only if dga_seed ≠ 0

Loader: Configuration & DGA

The loader configuration comes encrypted and contains the following information: C&C domain, port, user-agent and a seed for the Domain Generation Algorithm (DGA). In case seed is not zero, the loader uses a DGA method to generate its C&C domain, based on the seed and the calendar day of the communication. The configuration of the loader is decrypted using the following algorithm:

```
def decrypt_buf(buf):

k = 8

j = 5

for i in range(len(buf)):

xor_byte = (k + j) % 0xff

buf[i] = buf[i] ^ xor_byte

j = k

k = xor_byte
```

Configuration decryption algorithm

The DGA method is fully described in **Appendix B**.

Loader: C&C Communication

After getting the C&C domain, the loader contacts it to download the next and final stage of the infection chain. Although it sounds simple, the attackers operate the C&C server in a limited daily window, going online only for a few hours each day, making it harder to gain access to the advanced parts of the infection chain.

Loader: Next stage payload

At the next and final stage of the loader, the downloaded RAT is decrypted using a single byte XOR key, received from the C&C. Once the RAT's DLL is downloaded and decrypted, the DLL is loaded into the memory. The loader will then check the exported function against a hardcoded `djb2` hash value, and will call it upon a match.

Aria-body RAT analysis

The downloaded payload is a custom RAT dubbed **Aria-body**, based on the name given by the authors: `aria-body-dllX86.dll` .
Although the below analysis is of the 32bit variant malware, we have observed a 64bit variant as well, with similar functionality.

```
.rdata:1002D5AC      aAriaBodyDllx86 db 'aria-body-dllX86.dll'
.rdata:1002D5AC                                                ;
.rdata:1002D5C1      aAzmanager      db 'AzManager',0          ;
.rdata:1002D5CB      aDebugazmanager db 'DebugAzManager',0     ;
```

Strings found inside the "Aria-body" backdoor

The RAT includes rather common capabilities of a backdoor, including:

- Create/Delete Files/Directories
- Take a screenshot
- Search file
- Launch files using `ShellExecute`
- Enumerate process loaded modules
- Gather files' metadata
- Gather TCP and UDP table status listing
- Close a TCP session
- Collect OS information
- Verify location using `checkip.amazonaws.com`
- (Optional) Inter-process pipe based communication

Some of **Aria-body** variations also included other modules such as:

- USB data gathering module
- Keylogger module to collect raw input device-based keystrokes – added by February 2018
- Reverse socks proxy module – added by February 2018
- Loading extensions module – added by December 2019

All the supported functionality of the backdoor is described in the table of **Appendix A**.

Unique Characteristics

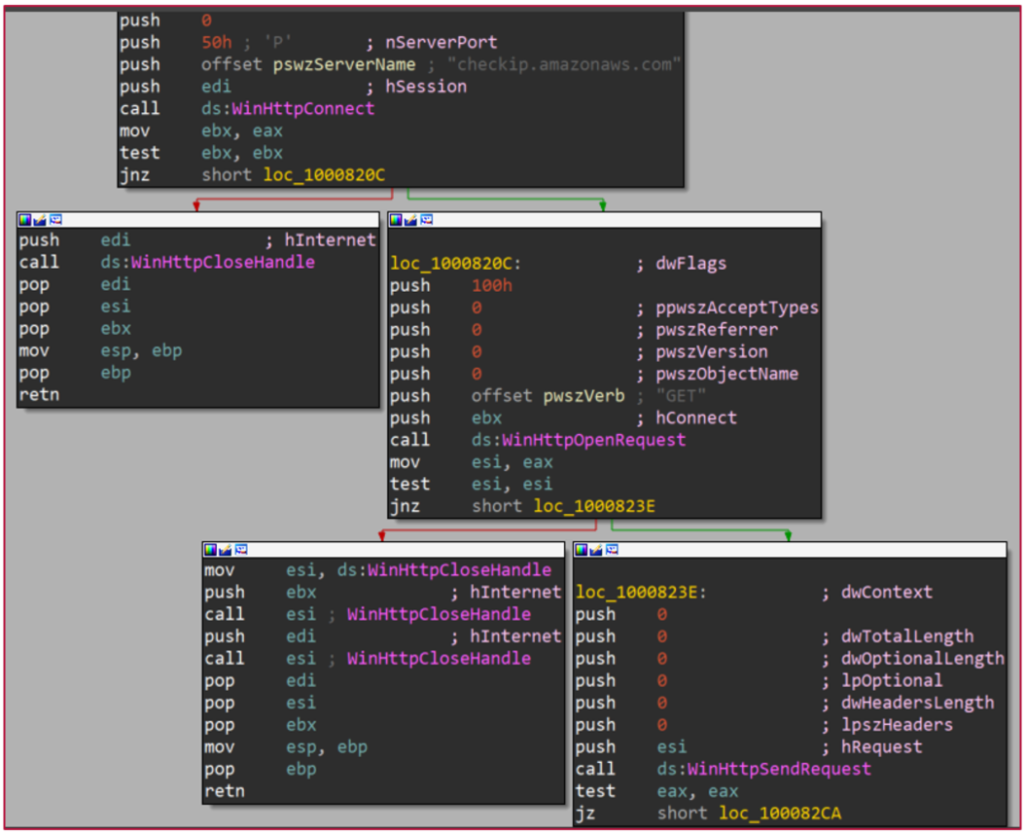
In the following section, we go over some of the techniques by which the backdoor was implemented, and highlight the characteristics that might help other researchers recognize this backdoor and correlate it with other samples.

Initialization

As previously mentioned, the backdoor contains an exported function, which the previous loader calls after loading the payload into the memory. Upon executing the backdoor, it initializes a struct named `MyDerived` and several structs used for HTTP and TCP connection.

Information Gathering

Aria-body starts with gathering data on the victim's machine, including:
Host-name, computer-name, username, domain name, windows version, processor ~MHz, MachineGuid, 64bit or not, and public IP (using `checkip.amazonaws.com`).

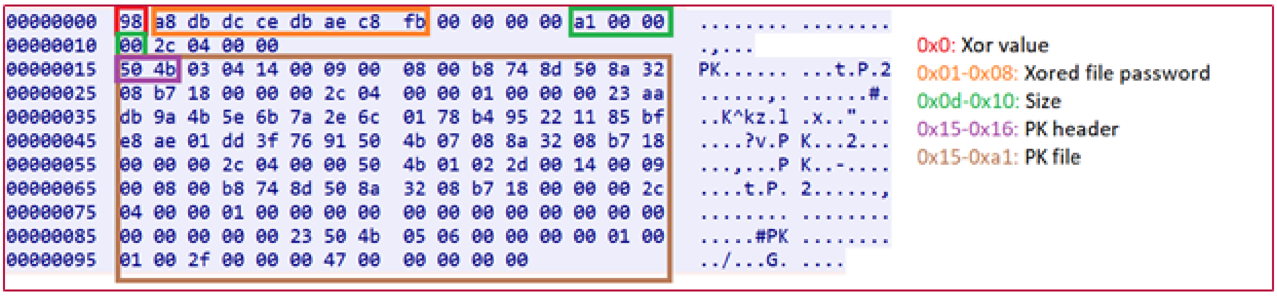


Aria-body using checkip.amazonaws.com service to get victim's IP

This data is gathered into an information structure which the RAT zips with an 8 bytes random generated password, which is then XORed with one byte.

C&C Communication

The communication to the C&C server is available by either HTTP or TCP protocols. The malware decides which protocol to use by a flag in the configuration of the loader. The collected data is sent to the C&C domain along with the XORed password, and the XOR key in the following format:



C&C communication structure

Whether the message is sent by TCP or HTTP, the payload format is the same. However, when HTTP is selected, the following GET request format is used:
`https://%s:%d/list.html?q=<random string>`

After the initial request to the C&C server, the backdoor then keeps listening to additional commands from the server. When a command is received, it is matched against a list of commands, and executed accordingly. A full list of supported commands is available in **Appendix A**.

The Outlook DLL Variant

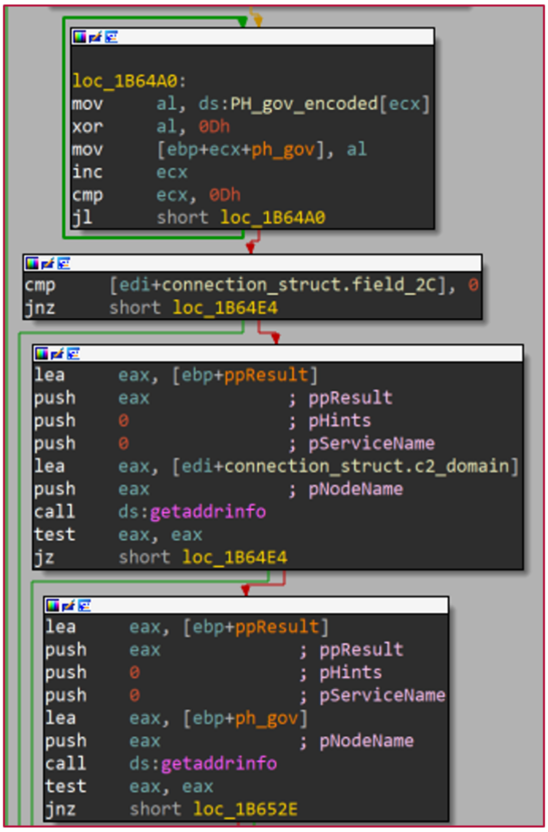
During our research we have found another, quite a unique variant of Aria-body, uploaded to VirusTotal from the **Philippines**. This variant's DLL was named `outllib.dll` , and it was part of a RAR archive named `Office.rar` . It utilized a DLL side-loading technique, abusing an old **Outlook** executable.

What was unusual in this variant was the fact that there has no loader as part of the infection chain, unlike all the other versions of Aria-body. As a result, it did not get any configuration from the loader, and included hardcoded configuration within it.

The payload has two different C&C domains:

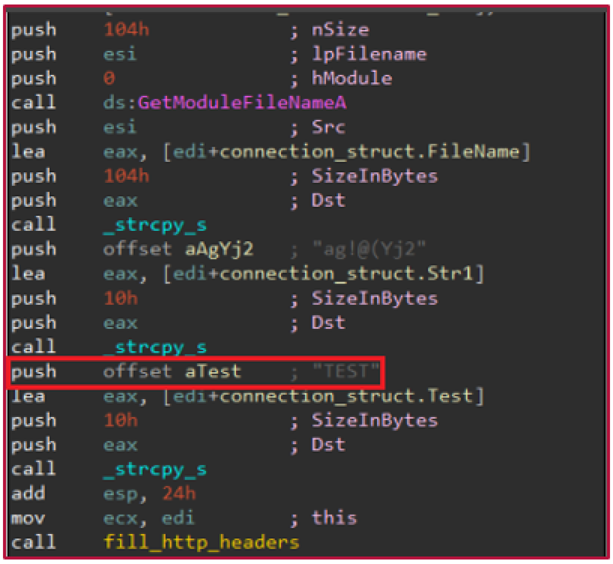
- `blog.toptogear[.]com` – which it gets by XORing an encrypted string with the byte `0x15` .
- `202.90.141[.]25` – an IP associated with a **Philippine government website**, which is being used in case that the first C&C domain cannot be resolved.

This variant also has some extra features that the main variant of **Aria-body** does not include, such as a USB-monitor module. On the other hand, this variant is missing the keylogger component and the reverse-socks module, observed with the main **Aria-body** variants. This evidence suggests that this is an out of scope variant of the backdoor, tailored for a specific operation.



Usage of Philippines govt' C&C server as backup

Moreover, we have seen that **Aria-body's** main variant has a version that was compiled sometime after `outlib.dll` variant was, and some strings within this variant could suggest that it was a test variant of this special version:



"TEST" string as part of the connection struct of "outllib.dll"

Finally, this version of Aria-body includes the following string:

`c:\users\bruce\desktop\20190813\arn\agents\verinfo.h` , with the "**ar**" in "**ar**n" possibly standing for "**Ar**ia".

Attribution

We were able to attribute our campaign to the Naikon APT group using several similarities we observed to the previously disclosed information about Naikon's activity by Kaspersky in 2015: 1, 2. In this original operation, the Naikon APT group utilized a backdoor against different government institutions in APAC.

Going forward, we will refer to the backdoor analyzed by Kaspersky as **XsFunction** due to **PDB** path found in one of its samples:
`g:\MyProjects\xsFunction\Release\DLL.pdb`

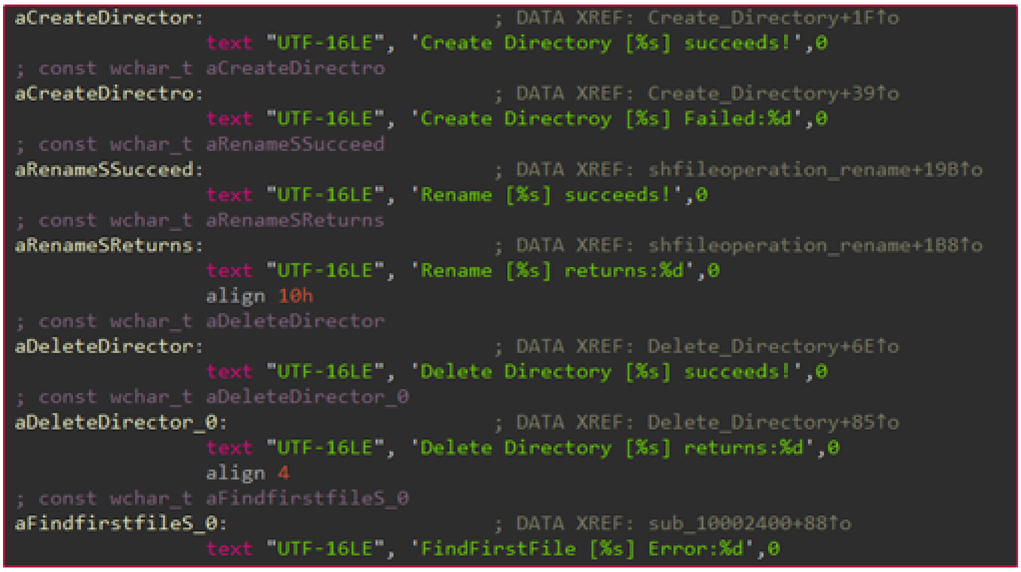
XsFunction is a full featured backdoor which supports 48 different commands. It allows the attacker to gain full control on the victim computer, perform file and process operations, shell commands execution, as well as to upload and download data and additional plugins.

We were able to find several similarities to previous operations (besides the obvious overlap in targeting), as well as specific similarities to the **XsFunction** backdoor.

String Similarity

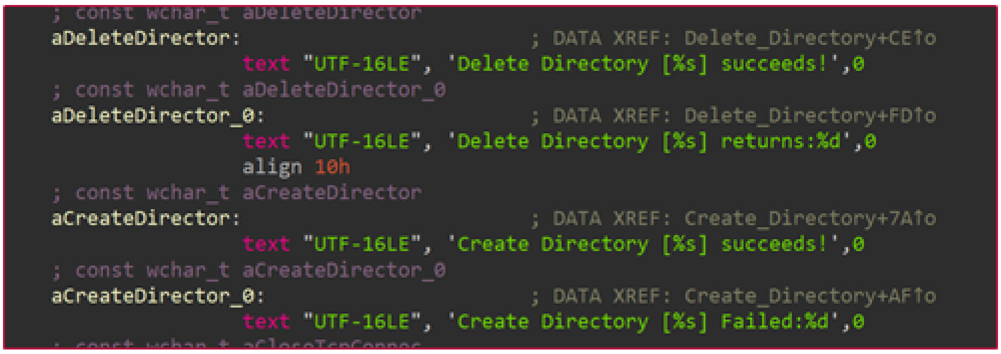
Aria-body backdoor has several **debug strings** that describe the functionality of the malware.

Some of these **exact** debug strings, can also be found in the **XsFunction** backdoor:



```
aCreateDirector:                        ; DATA XREF: Create_Directory+1F↑o
                text "UTF-16LE", 'Create Directory [%s] succeeds!',0
; const wchar_t aCreateDirectro
aCreateDirectro:                        ; DATA XREF: Create_Directory+39↑o
                text "UTF-16LE", 'Create Directroy [%s] Failed:%d',0
; const wchar_t aRenameSSucceed
aRenameSSucceed:                        ; DATA XREF: shfileoperation_rename+19B↑o
                text "UTF-16LE", 'Rename [%s] succeeds!',0
; const wchar_t aRenameSReturns
aRenameSReturns:                        ; DATA XREF: shfileoperation_rename+1B8↑o
                text "UTF-16LE", 'Rename [%s] returns:%d',0
                align 10h
; const wchar_t aDeleteDirector
aDeleteDirector:                        ; DATA XREF: Delete_Directory+6E↑o
                text "UTF-16LE", 'Delete Directory [%s] succeeds!',0
; const wchar_t aDeleteDirector_0
aDeleteDirector_0:                      ; DATA XREF: Delete_Directory+85↑o
                text "UTF-16LE", 'Delete Directory [%s] returns:%d',0
                align 4
; const wchar_t aFindfirstfileS_0
aFindfirstfileS_0:                      ; DATA XREF: sub_10002400+88↑o
                text "UTF-16LE", 'FindFirstFile [%s] Error:%d',0
```

Strings found in Aria-body backdoor



```
; const wchar_t aDeleteDirector
aDeleteDirector:                        ; DATA XREF: Delete_Directory+CE↑o
                text "UTF-16LE", 'Delete Directory [%s] succeeds!',0
; const wchar_t aDeleteDirector_0
aDeleteDirector_0:                      ; DATA XREF: Delete_Directory+FD↑o
                text "UTF-16LE", 'Delete Directory [%s] returns:%d',0
                align 10h
; const wchar_t aCreateDirector
aCreateDirector:                        ; DATA XREF: Create_Directory+7A↑o
                text "UTF-16LE", 'Create Directory [%s] succeeds!',0
; const wchar_t aCreateDirector_0
aCreateDirector_0:                      ; DATA XREF: Create_Directory+AF↑o
                text "UTF-16LE", 'Create Directory [%s] Failed:%d',0
```

Strings found in XsFunction (d085ba82824c1e61e93e113a705b8e9a)

Hashing Function Similarity

Both **XsFunction** and **Aria-body loaders** utilize the same hashing algorithm `djb2` to find which exported function should be run. In **XsFunction** the name of that function is `XS02` and in **Aria-body** it is `AzManager`.

```
push    7C8EB852h       ; "XS02" hash
mov     eax, [ebp+payMod]
push    eax
call    GetProcByHash   ; Manual getting of XS02 function address
```

XsFunction loader (Image by Kaspersky)

```
mov     edx, 2E9AD5FBh  ; AzManager
mov     ecx, ebx
call    search_func
test    eax, eax
```

Aria-body loader

Code Similarity

Some functions in the **Aria-body** backdoor are identical to functions used in the old **XsFunction** backdoor. One example is the function which gathers information about the installed software on the PC:

```
push    eax                     ; lpData
push    0                       ; lpType
push    0                       ; lpReserved
push    offset aDisplayname ; "DisplayName"
push    [esp+7D8h+hKey] ; hKey
call    edi ; RegQueryValueExW
test    eax, eax
jnz     loc_10005971

lea     eax, [esp+7C4h+cbData]
push    eax                     ; lpcbData
lea     eax, [esp+7C8h+var_678]
push    eax                     ; lpData
push    0                       ; lpType
push    0                       ; lpReserved
push    offset aDisplayversion ; "DisplayVersion"
push    [esp+7D8h+hKey] ; hKey
mov     [esp+7DCh+cbData], 40h ; '@'
call    edi ; RegQueryValueExW
lea     eax, [esp+7C4h+cbData]
push    eax                     ; lpcbData
lea     eax, [esp+7C8h+var_638]
push    eax                     ; lpData
push    0                       ; lpType
push    0                       ; lpReserved
push    offset aInstalldate ; "InstallDate"
push    [esp+7D8h+hKey] ; hKey
mov     [esp+7DCh+cbData], 18h
call    edi ; RegQueryValueExW
lea     eax, [esp+7C4h+cbData]
push    eax                     ; lpcbData
lea     eax, [esp+7C8h+var_620]
push    eax                     ; lpData
push    0                       ; lpType
push    0                       ; lpReserved
push    offset aUninstallstrin ; "UninstallString"
```

Aria-body information gathering

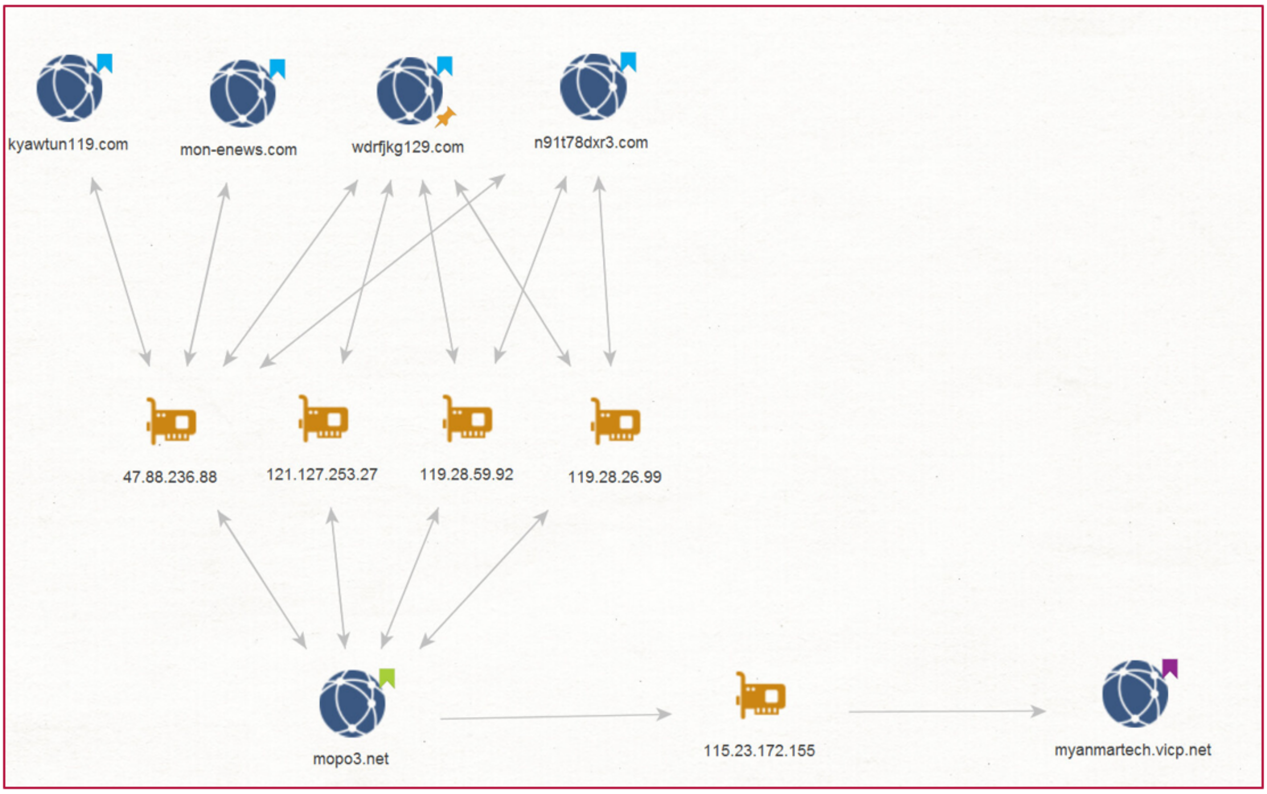
```
push    eax                     ; lpData
push    0                       ; lpType
push    0                       ; lpReserved
push    offset aDisplayname ; "DisplayName"
mov     ecx, [ebp+hKey]
push    ecx                     ; hKey
call    ds:RegQueryValueExW
mov     [ebp+var_21C], eax
cmp     [ebp+var_21C], 0
jnz     loc_10006962

mov     [ebp+cbData], 40h ; '@'
lea     edx, [ebp+cbData]
push    edx                     ; lpcbData
lea     eax, [ebp+var_690]
push    eax                     ; lpData
push    0                       ; lpType
push    0                       ; lpReserved
push    offset aDisplayversion ; "DisplayVersion"
mov     ecx, [ebp+hKey]
push    ecx                     ; hKey
call    ds:RegQueryValueExW
mov     [ebp+cbData], 18h
lea     edx, [ebp+cbData]
push    edx                     ; lpcbData
lea     eax, [ebp+var_650]
push    eax                     ; lpData
push    0                       ; lpType
push    0                       ; lpReserved
push    offset aInstalldate ; "InstallDate"
mov     ecx, [ebp+hKey]
push    ecx                     ; hKey
call    ds:RegQueryValueExW
mov     [ebp+cbData], 208h
lea     edx, [ebp+cbData]
push    edx                     ; lpcbData
lea     eax, [ebp+var_638]
push    eax                     ; lpData
push    0                       ; lpType
push    0                       ; lpReserved
push    offset aUninstallstrin ; "UninstallString"
```

XsFunction information gathering

Infrastructure overlap

Four of our C&C servers shared IPs with `mopo3[.]net` domain, this domain resolves to the same IP as the domain mentioned in Kaspersky's report: `myanmartech.vicp[.]net` .



Maltego – graph of infrastructure overlap

Conclusion

In this campaign, we uncovered the latest iteration of what seems to be a long-running Chinese-based operation against various government entities in APAC. This specific campaign leveraged both common toolsets like RoyalRoad RTF weaponizer, as well as a specially crafted backdoor named **Aria-body**.

While the Naikon APT group has kept under the radar for the past 5 years, it appears that they have not been idle. In fact, quite the opposite. By utilizing new server infrastructure, ever-changing loader variants, in-memory fileless loading, as well as a new backdoor – the Naikon APT group was able to prevent analysts from tracing their activity back to them.

Check Point SandBlast Agent protects against such APT attacks, and is capable of preventing them from the very first step.

Appendix A: Aria-body – Supported Commands

Command ID (Sent from C&C)	Sub Command ID (Sent from C&C)	Description	Command add date
0x1	0x0	Gather installed software's information	–
0x2	0x0	Get Disks information	–
0x2	0x1	File Search by name	–
0x2	0x2	Find Directory	–
0x2	0x4	Create Directory	–
0x2	0x6	SHFileOperaion – Delete Directory	–
0x2	0x7	SHFileOperaion – rename file	–
0x2	0x9	Delete File in a given path	–
0x2	0xa	ShellExecute 'open' command	–
0x2	0xb	ShellExecute 'open' command	–
0x2	0xe	Create new file and write its data	–
0x3	0x0	Get active processes information	–
0x3	0x2	Terminate Process	–
0x3	0x3	Get loaded modules information	–
0x4	all	Unique modules command: ARN – USB monitor module	only in outl-lib.dll variant
0x4	all	Unique modules command: aria-body – reverse socks proxy module	Feb 2018 – not in outllib.dll
0x5	0x0	Get MD5 of file	–
0x6	0x0	Get titles of running windows	–
0x6	0x1	Send WM_CLOSE message to given window name	–
0x7	0x0	Get TCP and UDP tables	–
0x7	0x1	Close given TCP connection	–
0x8	0x0	Start keylogger	Feb 2018 – not in outllib.dll

0x8	0x1	Stop keylogger	Feb 2018 – not in outllib.dll
0X9	0X0	Inject itself into rundll32.exe – spawn module	July 2018 – not in outllib.dll
0X9	0X1	Inject itself into rundll32.exe with UAC	July 2018 – not in outllib.dll
0X9	0X2	Inject itself to every process except explorer.exe	July 2018 – not in outllib.dll
0xa	0x1	Collect services data	Dec 2018 – not in outllib.dll
0xaa	0x1	Load extensions	Dec 2018 – not in outllib.dll
0xaa	0x2	'runas' with given process	–
0xaa	0x3	Zip-Directory	–
0xaa	0x4	Create Process and inject itself into it.	–
0xaa	0x5	UAC method (duplicate token from ntprint.exe)	–
0xaa	0x6	Send screenshot	–
0xaa	0x7	Send command to given extension	Dec 2018 – not in outllib.dll
0xaa	0x9	Destruction method	Dec 2018 – not in outllib.dll

Appendix B: DGA method

def DGA_method(seed_value):

domain = ""

tld = [".com", ".org", ".info"]

```
ta = time.localtime(time.time())

temp1 = math_s(ta.tm_year)

temp2 = math_s(dword(temp1 + ta.tm_mon + 0x11FDA))

temp3 = math_s(dword(temp2 + ta.tm_mday))

temp4 = math_s(dword(seed_value + temp3))

temp5 = math_s(dword(temp4 + 9))

length = (temp5 % 0xe) + 8

if length > 0:

for i in range(length):

temp6 = math_s(i + temp5)

domain += chr ( ( temp6 % 0x1a) + 0x61)

temp5 = math_s(dword(temp6 + 0xcdcdef))

domain += tld[temp6 % 3]

print(domain)
```

Appendix C: IOC list

Delivery:

MD5	SHA-1	SHA-256
f9d71f32de83f9ecfd-c77801a71da7bf	560423901a746055a4890c87d-abe2c2a59ee917a	d6841b2a82904efc52c6b0b9375d-dd3aa70de360c9f605341631358З
08428c94f45fb8f-f568a4a288778dfb7	00934d22f-b37b2de-f8276bc22ace5d-c950b66227	7df5442e5c334e-b81a2f871623fcbed859148223ef2(b0e628d02190d
5e37131cb-d756e10a9392d2280907592	c0c39b4ffe6fa7f-f627654fbd-d53a3bf638da4cb	6a8f59ad46ad22f272d5617e8d81(2abd5b162e3e9a9cc5dfb2f46ac

Aria-body loaders – 32bit

e9a23e084eb8cf95b70cde3afc94534b	96a918b4e54090c0294470c872c1b2075af1a
8561fa029f2158dc9932deee61febdac	3cecff13388d6ab45797ca2455caf5fd04ca9dd
31a4400789ae43b255464481320baa9e	1e3f303bbb35e709ff9d962c28c071656070aa
32b1916abff8bf0e7c51a2584c472451	513d99d714985ab53d75894357e4e87c69374
c2dc85559686575c268c8e97205b7578	b01d9454d84d04dd7a594dd2f899c77a40248
b779742b94b9265338c9b21f0cc88ba4	3f7190d530a98e157d799bdbe4fef8e69f1c50
ca3d5f02f453455f2b5522b8dceca658	0289a6db2fdda581b413768cd9318f33b5c005
bd1ef60ee835dd996ddcf4f22adaa142	1d7056e1bec6fadfba8b69d725e4a930bdb6fa
1dd0e12a886f3d1bded6e26f53592720	896e44af5a6f88c7be21d2f7225462f273f067f
07f724bdc662518ce6eac0ca723c929f	1eb758bcb0fc640835962aaa80199bdc867c7
dde75e82b665fc7d47cd870dae2db302	2f17d1f1766b2814d6347763c9ad94863e5bd
20cdf05867967642742d6b947ba71284	31cf5cb37d1d6e62add2cd4e59c2821a1a3c5
9b0cb194dd5e49ab6fbf490de42e6938	396c0c1dce196e9dc4e65aeb57d2bd1ec5e85
b8292fe24db8f86b11e6bf303c5f3ac5	69ea467bdfe5b7739553da7f93096a3ac9442
357a9f8268438d487303b267b26bde65	722b3dafbc14f8dce1048264451017d3f473f1a
40c49ecbe1b7bd0dbb935138661b6ca4	fe84b53aa8bb4e8ac3d2d9f86d2397d4a3cf5c
85e5d261c810e13e781f24505bb265ce	6a5a96f5637c898c0792ca9e76fc1854cf960d
77ea1eb5f6fd2605454764cd9b7ef62e	653aae2210a256a00ead6495e2c128d36d2e

ab260f3dc1ead01dfc6b7139d1eb983c	c2d3d9d7d7b64bbc6e522695105c31d5f1185
897994f378577ec1e09eaeb953cf603f	799ffe499b1a0d4b58ad9fa7b065b03432b96a
1f8f70afcd1a29920cb75e403bc590ff	441dfedf0583e799d2b37619316f8d924250d8
3d0320af4aeffa12660a3d4d8d6a5cf8	9dcf0be40d415c9cd86df39d608046a845b4a9

Aria-body loaders – 64bit

b65e38b86bd-d048638e17487a9c-ce181	6fbd039cbd-f2137a64390b80ba473949a3d-b5965	9033c75777e32c4014914272f7 1c152520dc204
97f3d2710d7b05f-da7e53bda3cdb-b3c8	088a603d6d144ab-b40145b6426acdad4b5813942	481a7868-effd2d356f85d9372d1ab5e35e9 da4

Aria-body payload

2ce4d68a120d76e703298f27073e1682	a84bde7bd58616e6f20ba106ca6e-f138e8cb6904
a8ee5b59d255a13172ec4704915a048b	48d4fe2ca8e4d71eaa8dead6bae629de47e-f77a7
e4f097ff8ce8877a6527170af955fc9b	4e76ad95cbfea448cb177c2de9c272141c11b8
537b21c71eb8381ed7d150576e3e8a48	be04013156a96ffb50646c5de1b9a1d7de99f0
43798a772bc4c841fc3f0b0aa157c1df	3223e64a1bfb25bc5ea95890ca438232adc-c7c35
c4397694368a0bfcb27ee91457878ef1	608f101efc89fbaf3aa7737b248a91c3d7540d9

Outlib.dll

63d64cd53f6-da3fd6c5065b2902a0162	09690a61e52716199 10a32efd-c91e756d0a6dc1e	f0e40b94e5e4ccbf94c94843dd1e-b8db21e36f5ec5d7e-f2a9512b026cef082e1

C&C Servers

realteks.gjdredj[.]com
spool.jtjewifyn[.]com
blog.toptogear[.]com
mon-enews[.]com
wdrfjkg129[.]com
n91t78dxr3[.]com
kyawtun119[.]com
www.ajtkgygth[.]com
news.nyhedmgtxck[.]com
dathktdga[.]com
www.rrgwmmwgk[.]com
dns.jmrmfitym[.]com
www.kyemtyjah[.]com
rad.geewkmy[.]com
cpc.mashresearchb[.]com
www.qisxnikm[.]com
dns.seekvibega[.]com
sugano.trictalmk[.]com
bbs.forcejoyt[.]com