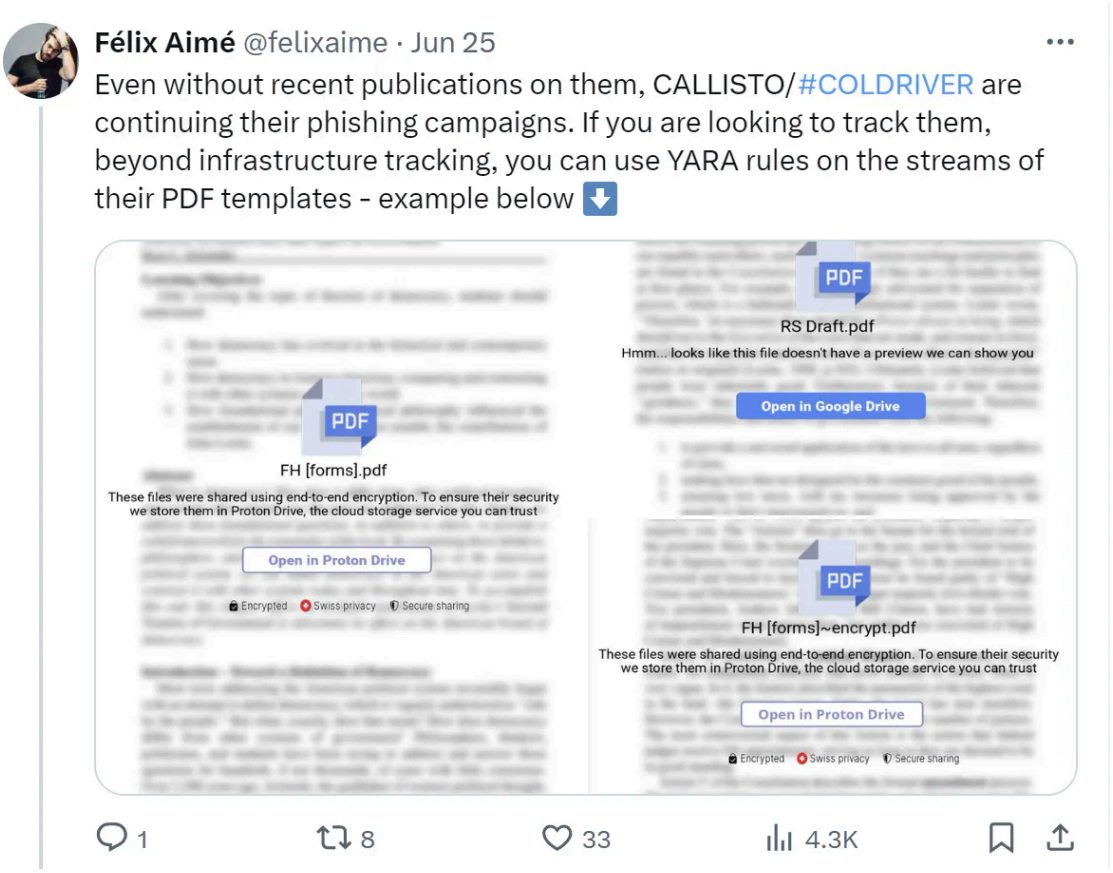
An interesting Callisto YARA rule

By David Cannings

Published: 2024-06-26 · Archived: 2026-04-05 12:51:19 UTC

Yesterday Félix Aimé posted on X (formerly known as Twitter) a neat YARA rule to detect PDF documents.



Tweet by @felixaime

These documents relate to the Callisto group, which is attributed to the Russian FSB by various Western intelligence agencies.

The rule makes use of YARA's `hash` module. But what is it hashing, and why does it work?

The rule

A text version of the initial rule is provided below - I added the author and description for clarity.

```
import "hash"

rule Calisto_PDF_streams {
  meta:
```

```
    author = "Félix Aimé @felixaime"
    description = "Detects Callisto PDFs"

  strings:
    $s1 = { 0A 73 74 72 65 61 6D 0A }          // <0x0a>stream<0x0a>
    $s2 = { 0A 65 6E 64 73 74 72 65 61 6D 0A } // <0x0a>endstream<0x0a>

  condition:
    uint32be(0) == 0x25504446 and
    for any i in (0..#s1) : (
      hash.md5(@s1[i]+8, @s2[i]-@s1[i]-8) == "b9950253cf88305a57cb350deb31c07e" or
      hash.md5(@s1[i]+8, @s2[i]-@s1[i]-8) == "9388a4ee5de0e59595a1e76aeb9796d5" or
      hash.md5(@s1[i]+8, @s2[i]-@s1[i]-8) == "9aa92ca954147945e6100ce345351d4c"
    )
}
```

Magic bytes

The first part of the rule `uint32be(0) == 0x25504446` checks that the file starts with the magic bytes `%PDF`.

```
00000000  25 50 44 46                                       |%PDF|
```

This is a useful optimisation in many circumstances as YARA can short-circuit and stop matching other conditions. See Florian Roth's optimisation guide for more on this topic.

However, the PDF magic does not need to appear at offset zero to open in most readers[1]. When generated by current software the magic should normally appear at the start of the file, but an easy way to break the rule would be to add random bytes at the start. When signaturing malicious documents it is often helpful to remove or tweak this condition.

The `for` loop

The statement `for any i in (0..#s1)` will loop from zero to the number of matches (see below for a comment on this).

The data for each stream in the document is surrounded by `stream` / `endstream`, which both have newlines. This part of the condition ensures we check each stream.

The `hash.md5` comparison

The final part of the rule compares chunks of the file to three specific hash values. The `md5` function takes an `offset` and `size`.

The offset is `@s1[i] + 8` - the `@` symbol in YARA returns the offset for the match. This will be the offset of the current `$s1` string, which matches `\x0astream\x0a`. Eight bytes are added to skip the start of stream marker.

The size is slightly more complicated. `@s2[i]` is the offset of the current end of stream marker. The address of the start of stream market is subtracted, along with 8 additional bytes.

```
 <0x0a>stream<0x0a>data data data<0x0a>endstream<0x0a>
^                ^            ^
|_ @s1           |_ @s1+8     |_ @s2
```

This provides the length of the data to hash. Note that if the offset of `@s2[i]` is less than `@s1[i]` the length would be negative, which is not ideal. We fix this below.

So what is being signatured?

In order to test further it is necessary to find a file. Fortunately one is available on VirusTotal[2] which hits the rule.

The condition can be modified to print the location of the match (it is necessary to add `import "console"` at the top of the rule for this to work correctly).

```
condition:
 uint32be(0) == 0x25504446 and
 for any i in (0.. math.max(math.min(#s1 - 1, #s2 - 1), 0) ) : (
   (hash.md5(@s1[i]+8, @s2[i]-@s1[i]-8) == "b9950253cf88305a57cb350deb31c07e" or
   hash.md5(@s1[i]+8, @s2[i]-@s1[i]-8) == "9388a4ee5de0e59595a1e76aeb9796d5" or
   hash.md5(@s1[i]+8, @s2[i]-@s1[i]-8) == "9aa92ca954147945e6100ce345351d4c") and
   console.log(@s1[i]) and console.log(hash.md5(@s1[i]+8, @s2[i]-@s1[i]-8))
 )
```

Running this prints the following:

```
 > yara64 .\rule.yar .\4f90aed9138ae23a32c2afce3237a891b50a39d04e169e79dd98f29e156295b2.hostile
179310
b9950253cf88305a57cb350deb31c07e
Calisto_PDF_streams .\4f90aed9138ae23a32c2afce3237a891b50a39d04e169e79dd98f29e156295b2.hostile
```

The address of the `$s1` match was printed, which tells us that the `stream` tag starts at offset 179,310 (0x2BC6E).

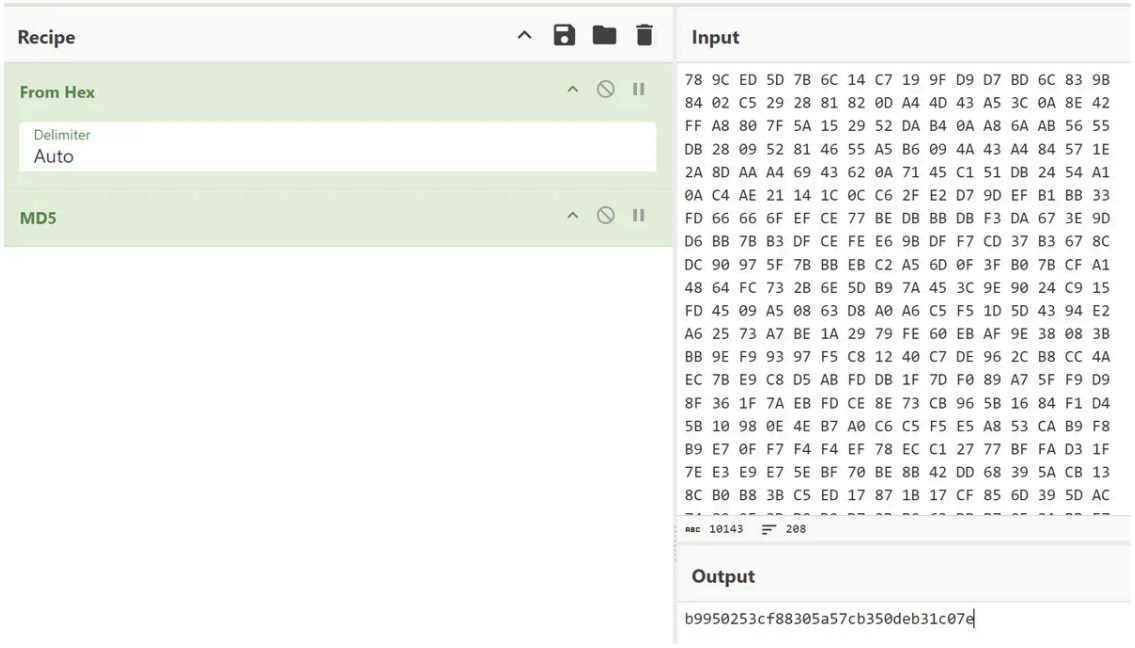
Inspecting the file with the PDF template for 010 Editor reveals an object that starts at offset 0x2BB86:

```
 00000000  34 33 20 30 20 6f 62 6a 0a 3c 3c 0a 2f 54 79 70  |43 0 obj.<<./Typ|
00000010  65 20 2f 58 4f 62 6a 65 63 74 0a 2f 53 75 62 74  |e /XObject./Subt|
00000020  79 70 65 20 2f 49 6d 61 67 65 0a 2f 57 69 64 74  |ype /Image./Widt|
00000030  68 20 39 36 0a 2f 48 65 69 67 68 74 20 39 36 0a  |h 96./Height 96.|
00000040  2f 43 6f 6c 6f 72 53 70 61 63 65 20 2f 44 65 76  |/ColorSpace /Dev|
00000050  69 63 65 52 47 42 0a 2f 42 69 74 73 50 65 72 43  |iceRGB./BitsPerC|
```

```
00000060  6f 6d 70 6f 6e 65 6e 74 20 38 0a 2f 46 69 6c 74  |omponent 8./Filt|
00000070  65 72 20 2f 46 6c 61 74 65 44 65 63 6f 64 65 0a  |er /FlateDecode.|
00000080  2f 44 65 63 6f 64 65 50 61 72 6d 73 20 3c 3c 0a  |/DecodeParms <<.|
00000090  2f 50 72 65 64 69 63 74 6f 72 20 31 35 0a 2f 43  |/Predictor 15./C|
000000a0  6f 6c 6f 72 73 20 33 0a 2f 42 69 74 73 50 65 72  |olors 3./BitsPer|
000000b0  43 6f 6d 70 6f 6e 65 6e 74 20 38 0a 2f 43 6f 6c  |Component 8./Col|
000000c0  75 6d 6e 73 20 39 36 0a 3e 3e 0a 2f 53 4d 61 73  |umns 96.>>./SMas|
000000d0  6b 20 34 34 20 30 20 52 0a 2f 4c 65 6e 67 74 68  |k 44 0 R./Length|
000000e0  20 33 33 31 32 0a 3e 3e 0a 73 74 72 65 61 6d 0a  | 3312.>>.stream.|
```

This is a 96x96 pixel image and the raw data is in `/FlateDecode` format[3].

To be absolutely certain we can copy the bytes between `stream` and `endstream` and hash them, confirming it matches the expected value.



CyberChef used to hash the stream data

The data for object 43 can also be extracted from the PDF, which ironically yields a PDF icon used by the threat actor.

We now know that the YARA rule is hashing all stream data in the PDF, looking for a specific 96x96 pixel image which is the icon above.

A second image matching a different MD5 hash can be extracted from another related document:

Performance and readability considerations

A few potential improvements to the rule are described below. Some are practically essential (e.g. if deploying to a streaming service like VirusTotal Live Hunt), others help with future maintenance.

Loop iterations

The first observation is that the rule tries to loop too many times. This happens because indexes start at zero, but the total count `#s1` starts at one. We could quickly fix this by subtracting one.

A second observation is that the count of `#s1` and `#s2` should be the same *in a well formed document*. However, it is theoretically possible that `<0x0a>stream` appears more than `<0x0a>endstream` (or vice versa). We should cap to the smallest number.

The final observation is that total loop iterations are unbound. A maximum should be applied here, for example the first fifty streams in the file.

The `math` module can be used to fix all of these. For example to hash a minimum of zero and a maximum of 50 streams:

```
math.max(math.min(math.min(#s1, #s2), 50) - 1, 0)
```

Reducing calls to `hash.md5(..)`

To minimise what we hash we can choose to call `hash.md5` only when the length is within a set range. This avoids hashing small objects or very large data which is unlikely to be relevant[4].

One question I had when inspecting this rule is: are multiple calls to `hash.md5(..)` inefficient? Fortunately Wes Shields (@wxs) confirmed the answer is no - multiple calls with the same input are cached[5].

Adding known data

In this example we know the image is 96x96 pixels. Therefore, adding more known good matches such as `/Height 96` could help to reduce the number of files which need to be checked.

Readability

Wes also noted that YARA supports string sets. The initial request for this feature was identical to our requirement here.

The condition can be rewritten like so, ensuring the rule does not repeatedly contain references to the hashing function.

```
for any stream_md5 in (
 "b9950253cf88305a57cb350deb31c07e",
 "9388a4ee5de0e59595a1e76aeb9796d5",
 "9aa92ca954147945e6100ce345351d4c"):
 ( // do stuff )
```

Optimising the rule

Taking the improvements suggested above we can update the rule like so:

```
 import "hash"
import "math"

rule Calisto_PDF_streams {
  meta:
    author = "Félix Aimé @felixaime, updated by David Cannings @edeca"
    description = "Detects Callisto PDFs"

  strings:
    $start = { 0A 73 74 72 65 61 6D 0A }        // <0x0a>stream<0x0a>
    $end = { 0A 65 6E 64 73 74 72 65 61 6D 0A } // <0x0a>endstream<0x0a>

    $known_1 = "/Height 96"
    $known_2 = "/Height 35"

  condition:
    uint32be(0) == 0x25504446 and
    any of ($known*) and
    for any stream_md5 in (
      "b9950253cf88305a57cb350deb31c07e",
      "9388a4ee5de0e59595a1e76aeb9796d5",
      "9aa92ca954147945e6100ce345351d4c"):
    (
      // Check the first 50 streams (maximum)
      for any i in (0.. math.max(math.min(math.min(#s1, #s2), 50) - 1, 0)) : (
        // Require a minimum of 1KiB
        @end[i] - @start[i] > 1024 and

        // Hash a maximum of 20KiB bytes
        @end[i] - @start[i] < 20480 and

        // Match to a known MD5 hash
        hash.md5(@start[i] + 8, @end[i] - @start[i] - 8) == stream_md5
      )
    )
}
```

Not all of the suggested improvements will work for every rule. In this case, the updated rule matches the same six files that can be obtained using a VirusTotal retrohunt for the original rule.

No sample files were found for one of the MD5 hashes. It is possible the additional checks will not detect some of the files the original author had available.

Conclusion

This is neat technique which - if employed sensibly - could be used to find interesting PDFs. There is plenty of stream data in a typical PDF ranging from text to images to fonts. Threat actors often reuse templates which yields plenty of opportunity for signatures.

However it is important to be aware of performance, lest you receive a dreaded email from somebody at VirusTotal saying your rule has been disabled 🙈.
