ML approach to RTF Detection

Matt Richard

Problem Statement

Malicious RTF Documents

- New 0-day every year from 2010-2016 using RTF
- Rich OLE model to exploit
- Current method has severe precision vs recall tradeoff
- Widely used in many different attacks
  - traditional espionage
  - dissidents
  - crimeware

Once upon a time, in 2009...

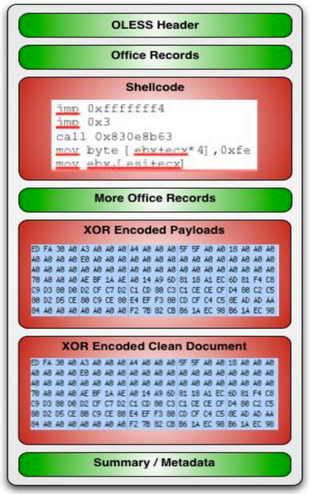
Raytheon
Customer Success Is Our Mission

Advanced Analysis of Malicious Documents

Matt Richard

Office Exploit Structure

- Everything included
- Permutations
  - Clean document
  - Trojan payload
  - Shellcode
  - Obfuscations



PDF 0-day 7/15/2009

- Adobe > v9 include Flash
- Flash is a programming language
  - Perfect for implementing heap sprays
- PDF loads heapspray SWF
- PDF loads exploit SWF
- Payload at fixed offset
- Two payloads
- First payload launches clean PDF
- Second payload installs backdoor

Tools - Yara

- http://code.google.com/p/yara-project/
- Classification
- Windows/*NIX
- No data transforms
- Non-linear scan times
- Simple and correlated rules
  - Ascii, binary, regex, wildcards

```
Rule PDF_Flash_Exploit
{
strings:
   $a = "%PDF-1."
   $j = "(pop\\056swf)"
   $k = "(pushpro\\056swf)"
   $b = "(         a.swf)"
condition:
   ($a at 0) and ($j or $k or $b)
}
```

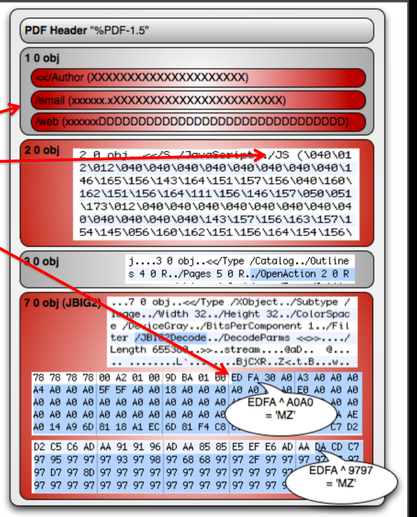
PPT Case Study – Using Yara

```
[matt@localhost]$ yara -s docfiles.yara 1d65f1be33c9d72030508c3df24ec780

EXPERIMENTAL_shellcode_lodsb_xor_stosb_decode
000018EC: EB EA FC 33 C9 EB 01
000018DE: C0 C8 06 75 03 74 01 E8 32 C3 AA 49

HIGH_xor_e
0
```

```
rule EXPERIMENTAL_shellcode_lodsb_xor_stosb_decode
{
strings:
$a = { c0 c8 06 75 03 74 01 ?? 32 c3 aa 49 }
$b = { eb ea fc 33 c9 eb 01 }
condition:
any of them
}
```

JBIG2 PDFs

- First JBIG2 1/15/2009
  - Unique metadata
  - JS Ocal Encoding
  - 1 or 2 Payloads
  - XOR 0xa0 / 0x97



Motivation

- There is a lot of ML FUD
- ML approaches seem “hard” to do
- Without intuition about how it works, it doesn’t benefit analysts
- Create detection with high recall ***and*** high precision

- Not the first ML for detection talk, but the first:
  - In plain english
  - Without a ROC curve
- See PDFRate for a good example of other use
  - https://csmutz.com/pdfrate/

PDFrate

A machine learning based classifier operating on document metadata and structure

{\rtf1
{\obj}\ansi\deff0{\fonttbl{\f0\fnil\fcharset0 Calibri;}}
{*\generator Msftedit 5.41.21.2510;}\viewkind4\uc1\pard\sa200\sl276\slmult1\lang9\f0\fs22
Reaching a deal on Iran's nuclear program is a matter of political will, and the "security of the world is at stake," the European Union's top diplomat said Sunday.\par
\par
European Union High Representative Federica Mogherini made the comments in Vienna, Austria, ahead of a meeting with U.S. Secretary of State John Kerry.\par
\par
The U.S. and its negotiating partners Britain, France, Germany, Russia and China -- known as the P5+1 -- as well as the European Union hope to reach a comprehensive nuclear deal with Iran before June 30.\par
\par
\par
\par
{\object\objocx\f37\objsetsize\objw1440\objh1440{*\objclass Forms.Image.1}{*\objdata
01050000020000000e000000466f726d732e496d6167652e3100000000000000000000480000
d0cf11e0a1b11ae10000000000000000000000000000000000003e000300feff090006000000000000000000000001000

ffffffffffffffffff0000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000105000000000000}}!!!!ñ◆qÕ<NUL>Ò@\9◆BÒ,+ñ◆À◆×F
wà¢◆^U'¶◆¾^A^Z^NxÑº◆Ç"¹ØmU2^DeZS?ê^S[^BpzùÒ¤ú^VÁz:◆Gpì$þeòqY*<FS>I◆<CR>ïné[ªN<FS>9Ý¶<◆gÚ^Aj"ÕL
¼◆×«·mÏ÷0◆Ì◆D9§^N¾Êé°@◆Íæñ◆◆<FS>ú^D,◆°◆◆\e,+vÃ-@\ñ±*>◆@^A%þF^Yû^N¾û◆<BS>AÓ^P×«¿mÏ÷0<NUL>I8µ<ES

```
=============================================================================
File: 'Iran_nuclear_talks.rtf' - size: 367631 bytes
---+----------+-------------------------------+---------------------------
id |index     |OLE Object                     |OLE Package
---+----------+-------------------------------+---------------------------
0  |00002425h |format_id: 2                   |Not an OLE Package
   |          |class name: 'Forms.Image.1'    |
   |          |data size: 18432               |
---+----------+-------------------------------+---------------------------
1  |00015B00h |format_id: 2                   |Not an OLE Package
   |          |class name:                    |
   |          |'Control.TaskSymbol.1'         |
   |          |data size: 3584                |
---+----------+-------------------------------+---------------------------
2  |00018753h |format_id: 2                   |Not an OLE Package
   |          |class name: 'Forms.Image.1'    |
   |          |data size: 3072                |
---+----------+-------------------------------+---------------------------
```

Precision vs Recall

- We write “precise” rules to minimize false positives
- We write “broad” (recall) rules to find new things we didn’t know about
- We frequently trade precision for recall
  - worse - sacrifice both for time

Adobe Flash Zero-Day Under Attack

By Ryan Naraine on May 10, 2016

A zero-day vulnerability in Adobe's ubiquitous Flash Player software is being exploited to launch malware attacks, the company warned in an advisory issued today.

The vulnerability, rated critical, will not be patched until May 12th.

The company credits Genwei Jiang of FireEye, Inc. with discovering the flaw, which provides an indication that it is being used in targeted attacks.

According to Adobe, the vulnerability is present in Windows, Mac OS X, Linux and Chrome OS.

From the **Adobe advisory**:

> A critical vulnerability (CVE-2016-4117) exists in Adobe Flash Player 21.0.0.226 and earlier versions for Windows, Macintosh, Linux, and Chrome OS. Successful exploitation could cause a crash and potentially allow an attacker to take control of the affected system.
>
> Adobe is aware of a report that an exploit for CVE-2016-4117 exists in the wild. Adobe will address this vulnerability in our monthly security update, which will be available as early as May 12.

```
File name iMarcRFP.doc
MD5 50f77cd868f6804e9a3bd1b0745ba36c
Submitted to VT on 2016-05-08 12:54:15

Hosted at hXXp://srv602.ddns.net/iMarcRFP.doc
SWF compile <dc:date>Mar 18, 2016</dc:date>
RTF time Revision time 2016-03-18 22:27:00
```

936082468662d4cdb63d5c20229be7c5 97475 saudireport2.doc
- RTF doc with embedded FWS

- contains an embedded office doc 021951c4dfce562ea17849d57933f14f
    - 3 objects to load the flash object - ocxname, objinfo and contents (flash file)
    - Flash file '<dc:date>Mar 18, 2016</dc:date>'
    - flash vars
        - Go_var1
        - flash2
        - var1

What works? What doesn’t?

```
rule flash_cve_2016_4117_rtf_exploit : rtf cve_2016_4117 exploit
{
meta:
  author = "mjr@fb.com"
  date = "2016-05-11"
  description = "Looks for flash invocation used by CVE-2016-4117 in RTF documents"
  sample = "936082468662d4cdb63d5c20229be7c5"
  reference = "http://www.securityweek.com/adobe-flash-zero-day-under-attack"
  confidence = 75
  severity = 8
strings:
  $rtf = "{\\rt"
  $object = "{\\*\\objdata"
  $flash_activex = "53686f636b77617665466c6173682e53686f636b77617665466c6173682e3231"
  $fufu = "66556655"
condition:
  $rtf at 0 and $object and $flash_activex and $fufu
}
```

I’m sold, show me how to do it

RTF ML Recipe

DIYable

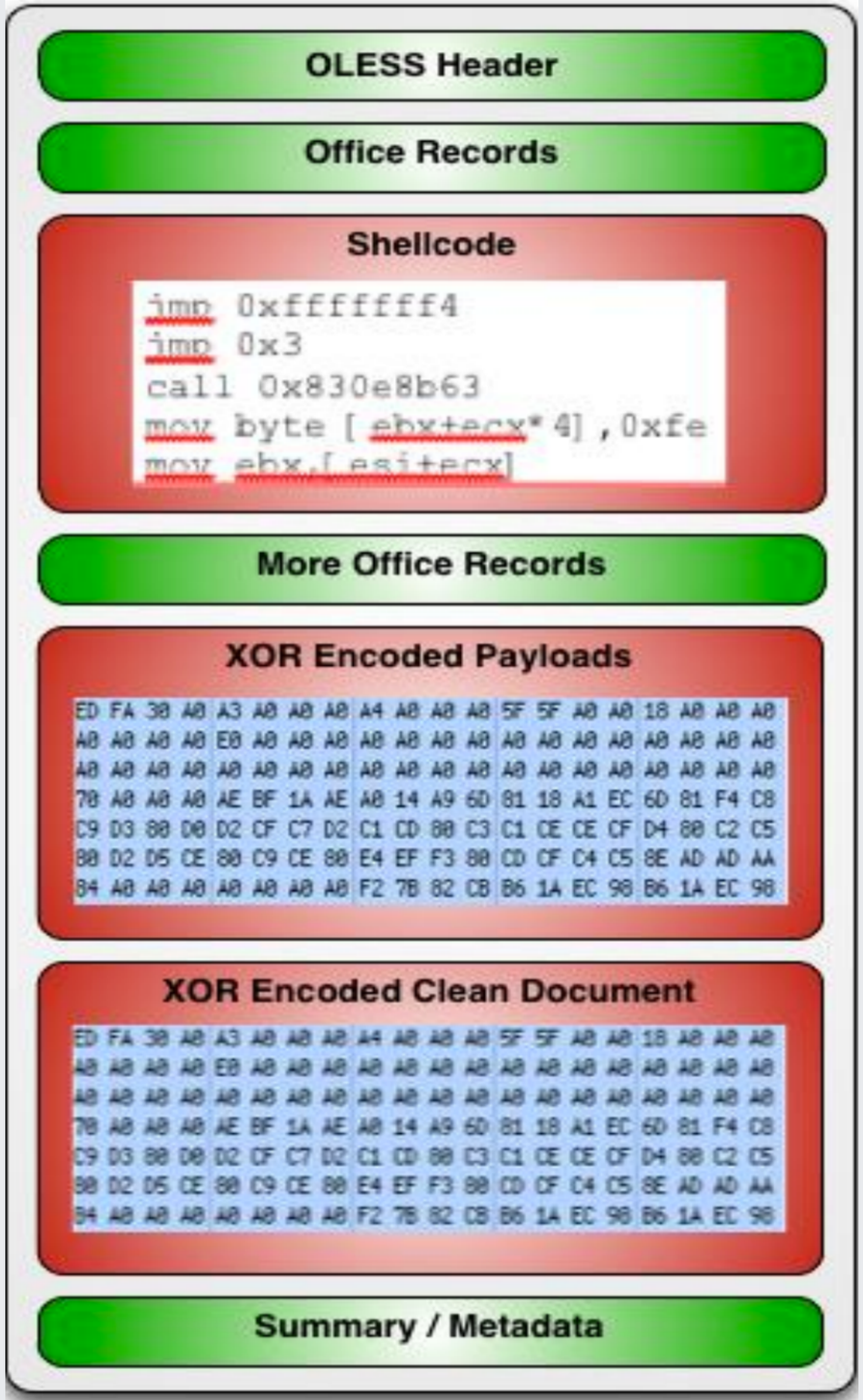
- Capture Intuition
- Identify Features
- Create a feature extractor
- Ground truth label data
- Pick an algorithm
- Train a model
- Predict and interpret

Capture Intuition

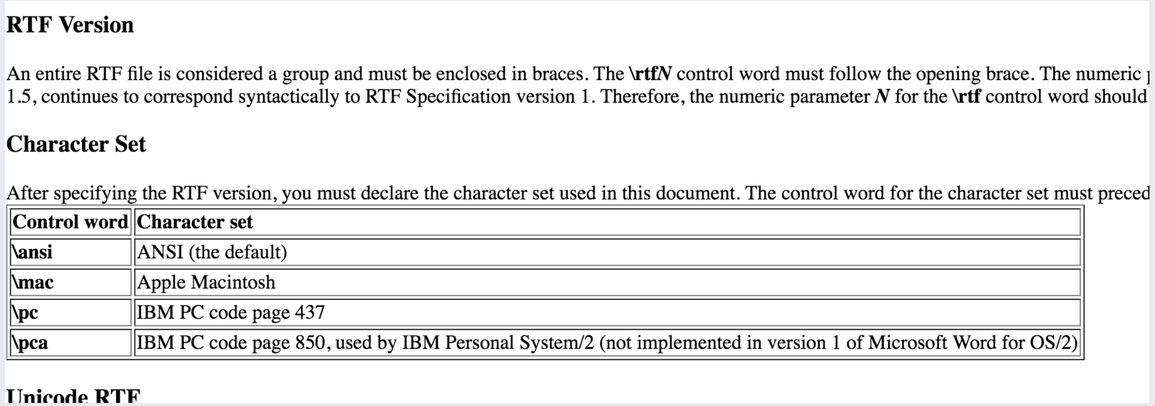
- Hypothesis - to be malicious an RTF:
  - contains content not normally found
  - mal content changes flow
  - when opened leads to code exec*

- Hypothesis - benign documents are almost exclusively made by tools, the tools create consistent structures

- ***A malicious and non-malicious document have different structures***



Identify Features



RTF Version

An entire RTF file is considered a group and must be enclosed in braces. The **\rtf*N*** control word must follow the opening brace. The numeric
1.5, continues to correspond syntactically to RTF Specification version 1. Therefore, the numeric parameter ***N*** for the **\rtf** control word should

Character Set

After specifying the RTF version, you must declare the character set used in this document. The control word for the character set must preced

Control word	Character set
\ansi	ANSI (the default)
\mac	Apple Macintosh
\pc	IBM PC code page 437
\pca	IBM PC code page 850, used by IBM Personal System/2 (not implemented in version 1 of Microsoft Word for OS/2)

Unicode RTF

- We theorize that to be malicious certain structures exist
  - objects
  - extra data
  - obfuscation

- Count and measure location of document structure
  - RTF Spec

Making Features (Binary)

```
rule rtf_small
{
condition:
  filesize < 500
}
```

Making Features

```
rule rtf_non_v1
{
strings:
  $a = /{\\rtf[0,2-9]/
condition:
  $a
}

rule rtf_v1
{
strings:
  $a = "{\\rtf1"
condition:
  $a
}
```

Making Features

```
rule rtf_high_binary_run
{
strings:
  $a = /\}.+[^\x07-\x7f]{10,}/
condition:
  $a
}

rule rtf_long_whitespace
{
strings:
  $a = /\{\\\*\\.+[\x0d\x0a\x07\x20\x09\x7e\x2a]{10,}/
condition:
  $a
}
```

Counting Features

```
rule rtf_ole_object
{
meta:
  type = "feature"
strings:
  $a = "{\\*\\datastore"
condition:
  $a
}
```

Create a feature extractor

```
[matt@localhost]$ yara -s docfiles.yara 1d65f1be33c9d72030508c3df24ec780

EXPERIMENTAL_shellcode_lodsb_xor_stosb_decode
000018EC: EB EA FC 33 C9 EB 01
000018DE: C0 C8 06 75 03 74 01 E8 32 C3 AA 49
```

By default yara tells us about matches and offsets

It does not tell us when it misses

We need feature vectors with count and offset

- sample1 = [1,0,1,1,1,0,1,1,0]
- sample2 = [1,1,0,0,0,1,1,1,1]

```
class YaraScan(object):
    results = {}
    data_len = 0

    def __init__(self, rulepath, results_prefix='yara', offset_postfix='offset'):
        self.engine = yara.compile(rulepath)
        self.results_prefix = results_prefix
        self.offset_postfix = offset_postfix

    def offset_ratio(self, strings):
        return min([b[0] for b in strings]) / float(self.data_len)

    def callback(self, data):
        rule = "%s_%s" % (self.results_prefix, data['rule'])
        offset_name = "%s_%s" % (rule, self.offset_postfix)
        if data['matches'] == True:
            match_len = len(data['strings'])
            self.results[rule] = 1 if match_len == 0 else match_len
            self.results[offset_name] = 0 if match_len == 0 else self.offset_ratio(data['strings'])
        else:
            self.results[rule] = 0
            self.results[offset_name] = 0
        yara.CALLBACK_CONTINUE

    def scan_file(self, filename):
        with open(filename, 'rb') as f:
            data = f.read()
        return self.scan_data(data)

    def scan_data(self, data):
        self.data_len = len(data)
        self.reults = {}
        self.engine.match(data=data, callback=self.callback)
        return self.results
```

Yara Feature Vectors

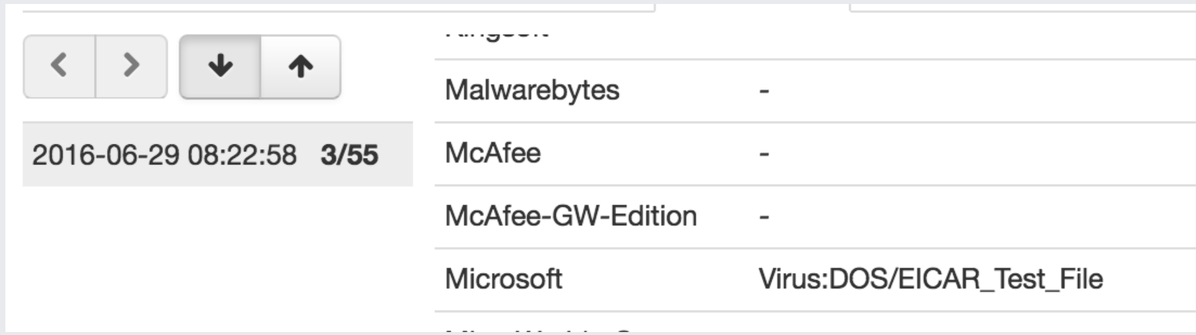

```
{'yara_rtf_ancalog': 0,
 'yara_rtf_ancalog_offset': 0,
 'yara_rtf_ansi_char': 3,
 'yara_rtf_ansi_char_offset': 3.8081663407057 62e-05,
 'yara_rtf_author': 1,
 'yara_rtf_author_offset': 0.23472177264702923,
 'yara_rtf_background': 0,
 'yara_rtf_background_offset': 0,
 'yara_rtf_nofeaturethrottle1': 1,
 'yara_rtf_nofeaturethrottle1_offset':
```

Ground Truth Labels

This is hard!

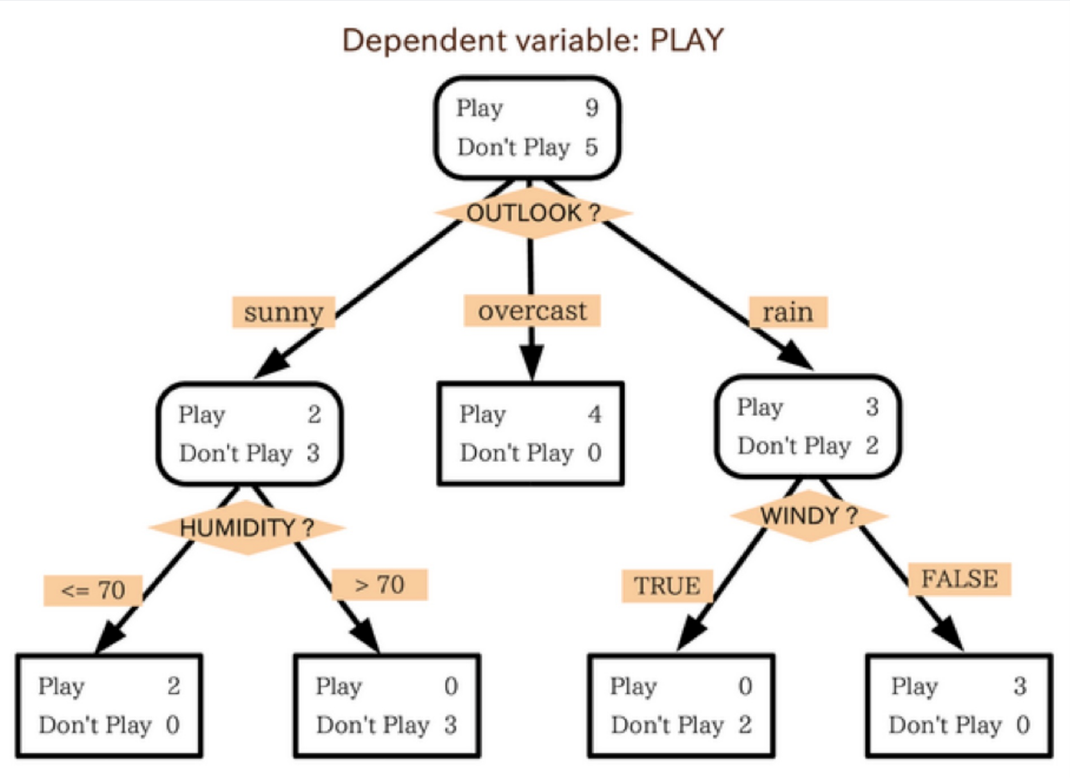
- We need some “ground truth” to train our algorithm
- Trust a system or do analysis?
- What is malicious?

```
labels = {
    'c13655fee08417cffa04d1bf71af4ad1': 'malicious',
    'eed9634170218760 40f24de7d2934c54': 'benign',
    '3f44a0f1d746cb99ab0321e73133ecae': 'malicious',
    '1adf29fb9b6c6ed972f1b9609b321314': 'malicious',
    '7faae63f2acff84420aea4c810be52b6': 'malicious',
    'eed9634170218760 40f24de7d2934c54': 'benign',
    '3aec93120e254c20993efe8df1bdb0b7': 'benign',
    '76cbcb6bcae46629a694db17fe2815fb': 'benign',
    'b544bdb59c4f5c6df815fa02358daafe': 'benign',
```



Pick an algorithm

- RandomForest
- Small labeled dataset, mimic analyst



rule	weight
yara_rtf_ole_control_object_offset	0.084104
yara_rtf_ole_control_object	0.067022
yara_rtf_ansi_char_offset	0.054197
yara_rtf_objdata_offset	0.044908
yara_rtf_empty_word_offset	0.043165
yara_rtf_high_binary_run	0.037604
yara_rtf_empty_word	0.037285
yara_rtf_objdata	0.036719
yara_rtf_ansi_code_page_char_offset	0.029801
yara_rtf_ansi_char	0.027635
yara_rtf_invalid_control_tag_offset	0.024327
yara_rtf_colortable_offset	0.022953
yara_rtf_invalid_control_tag	0.022599
yara_rtf_viewkind_offset	0.020068
yara_rtf_high_binary_run_offset	0.017897
yara_rtf_ansi_code_page_char	0.016537
yara_rtf_sv_offset	0.014489
yara_rtf_listviewctl_object_offset	0.014378
yara_rtf_word_class	0.013871
yara_rtf_embedded_ole_offset	0.012669

Train Model

- Success relies on:
  - good ground truth, balanced training data

```
forest = RandomForestClassifier(n_estimators = 1000)
train_features = s2.ix[label_cols, features]
train_labels = s2[label_cols].label
X_train, X_test, y_train, y_test = train_test_split(train_features, train_labels, test_size=0.25, random_state=0)
forest = forest.fit(X_train, y_train)
```

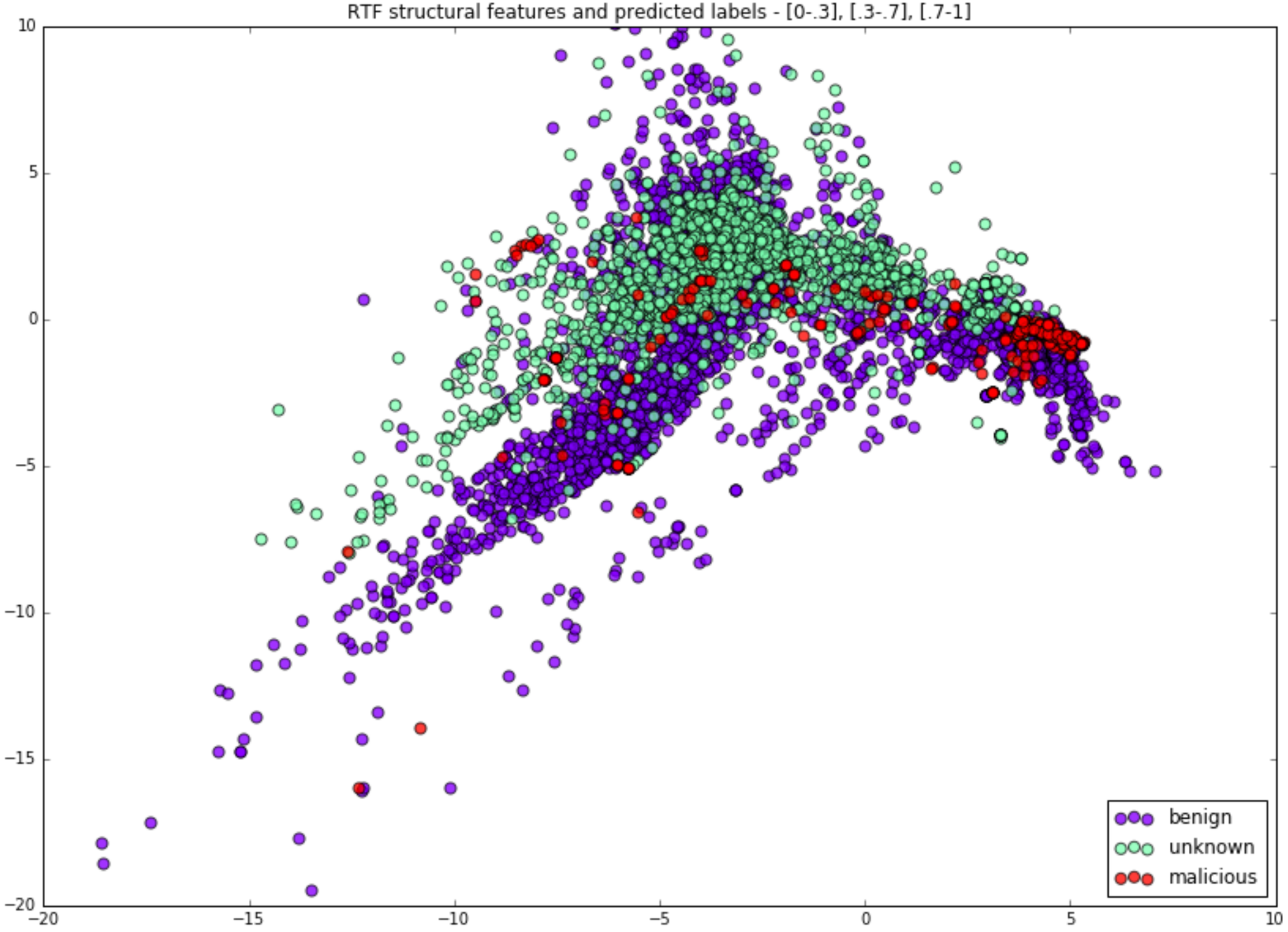
```
precision_recall_fscore_support(y_test, forest.predict(X_test))
```

```
(array([ 0.85714286,  0.96428571]),
 array([ 0.92307692,  0.93103448]),
 array([ 0.88888889,  0.94736842]),
 array([13, 29]))
```

Predict and Interpret

- Once we have trained we can predict arbitrary samples
- I used email, production and virustotal data to predict
- Surfaces interesting things as malicious and unknown

```
s2['prediction'] = forest.predict(s2.ix[:, features])
probs = pd.DataFrame(forest.predict_proba(s2.ix[:, features]), columns=forest.classes_)
s2['benign'] = probs.benign.tolist()
s2['malicious'] = probs.malicious.tolist()
bins = [0, 0.3, 0.7, 1]
bin_names = ['benign', 'unknown', 'malicious']
s2['threshold_label'] = pd.cut(s2['malicious'], bins, labels=bin_names, include_lowest=True)
```


RTF structural features and predicted labels - [0-.3], [.3-.7], [.7-1]
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
benign
unknown
malicious

Flash 0-day validation

- How does the model do against Mal RTF that we haven't trained on?

```
md5 = '93843bc95f91361305bfbbe57955a6fd'
y = YaraScan()
sample_features = pd.DataFrame([get_initial_features(path+sample_prefix+md5, y)])
probs = forest.predict_proba(sample_features[features])
vt_pos = get_vt_pos(md5)
if not vt_pos: vt_pos = 0
print "MD5: %s\nProb Mal: %02f\nProb Benign: %02f\nVT Pos: %d" % (md5, probs[0][1], probs[0][0], vt_pos)
```

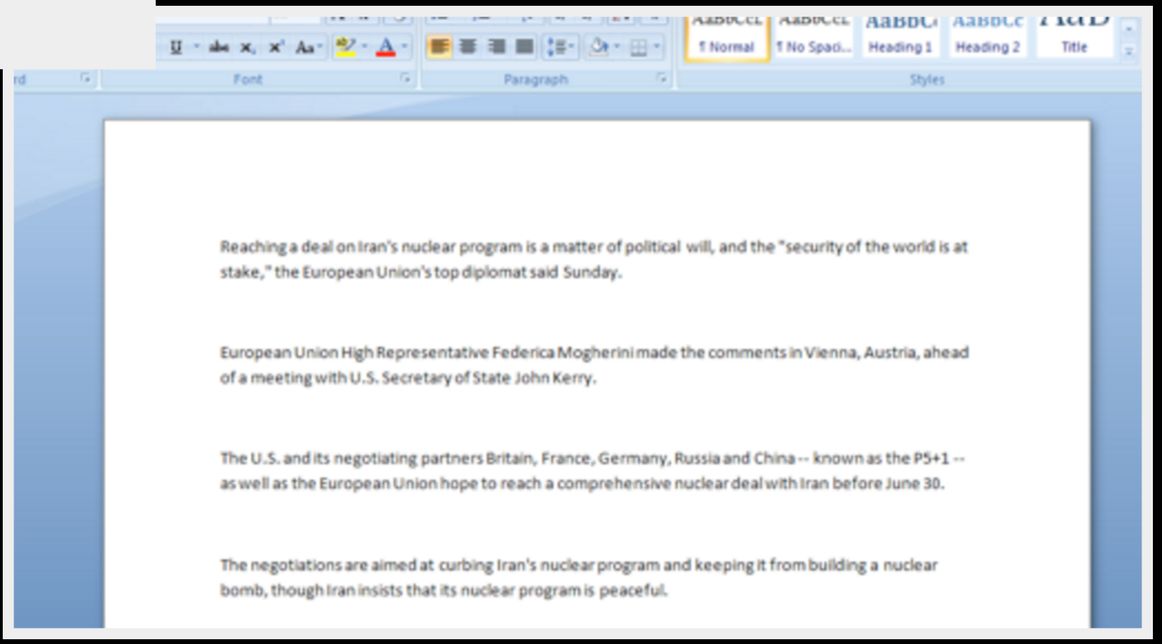
```
MD5: 93843bc95f91361305bfbbe57955a6fd
Prob Mal: 0.956000
Prob Benign: 0.044000
VT Pos: 0
```

Tsar Team Microsoft Office Zero Day CVE-2015-2424

Identification | Details | Content | Analyses | Submissions | ITW | Comments

2015-07-15 18:58:54 **1/56**
2015-06-30 10:44:29 **1/55**

SUPERAntiSpyware	-	5.6.0.1032	20150630
Symantec	-	20141.2.0.56	20150630
Tencent	-	1.0.0.1	20150630
TheHacker	-	6.8.0.5.584	20150630
TrendMicro	HEUR_RTFEXP.A	9.740.0.1012	20150630



<code></code>
<code></code>

```
md5 = "112c64f7c07a959a1cbff6621850a4ad"
y = YaraScan()
sample_features = pd.DataFrame([get_initial_features(path+sample_prefix+md5, y)])
probs = forest.predict_proba(sample_features[features])
vt_pos = get_vt_pos(md5)
if not vt_pos: vt_pos = 0
print "MD5: %s\nProb Mal: %02f\nProb Benign: %02f\nVT Pos: %d" % (md5, probs[0][1], probs[0][0], vt_pos)
```

```
MD5: 112c64f7c07a959a1cbff6621850a4ad
Prob Mal: 0.814000
Prob Benign: 0.186000
VT Pos: 33
```

Some Things I learned

- Many documents have OLE objects of many types
  - CorelDraw, Equation, etc
- Training on VT is “easy”, real world is harder
- Defining “bad” is harder than it looks
- Much easier if we had RTF analysis tools
  - nothing productive in CRITS, Laika, oletools

Recap

- Want to try this? - get the iPython Notebook
- Want to deploy this in production? - build it in python
- Want to help make this easier? CRITs or something else

Next Steps

- Going live in production
- Integrate flows with CRITs
- Build a proper analysis tool / service

Questions?