

CrowdSource: Applying machine learning to web technical documents to automatically identify malware capabilities

Joshua Saxe, Rafael Turner, Kristina Blokhin, Jose Nazario

Invincea Labs

A DARPA Cyber Fast Track research effort

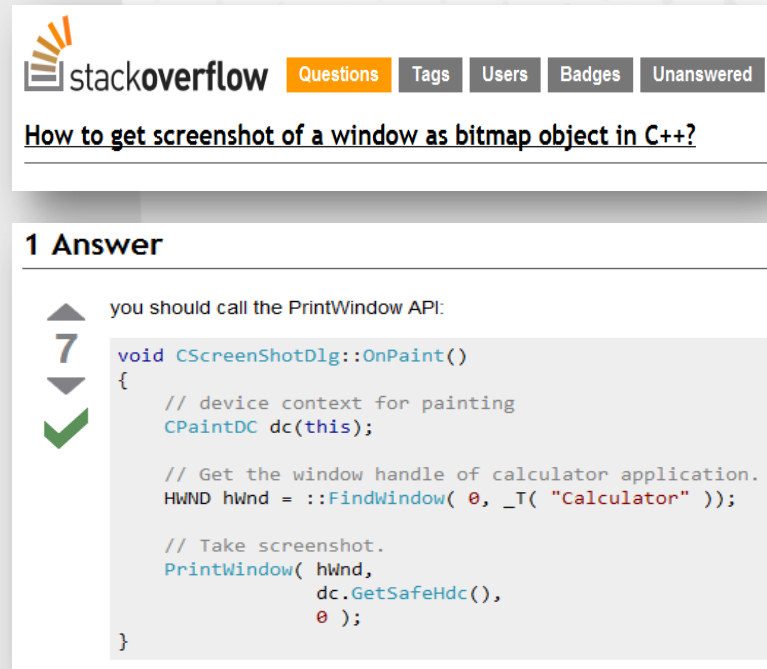
- The Internet is rife with text that combines example code with natural language description of its functionality
 - Why not use this data to train machine learning models to automatically reverse engineer software?
- Such an approach harnesses the web “crowd,” which holds more knowledge than the mind of any one malware reverse engineer
- As the web changes such an approach would automatically stay up to date with the latest programming idioms and APIs

KEY RESEARCH QUESTIONS

What judgments can we make about the capability profile of a malware sample based on this entirely automatic approach?

How does this approach compare with systems that rely on explicit encodings of expert knowledge to automatically analyze malware?

Typical web technical document:



The screenshot shows a Stack Overflow page. At the top, there is a navigation bar with 'Questions', 'Tags', 'Users', 'Badges', and 'Unanswered'. The question title is 'How to get screenshot of a window as bitmap object in C++?'. Below the question, there is one answer with a score of 7 and a green checkmark icon. The answer text says 'you should call the PrintWindow API:' followed by a code block in C++.

```
void CScreenShotDlg::OnPaint()
{
    // device context for painting
    CPaintDC dc(this);

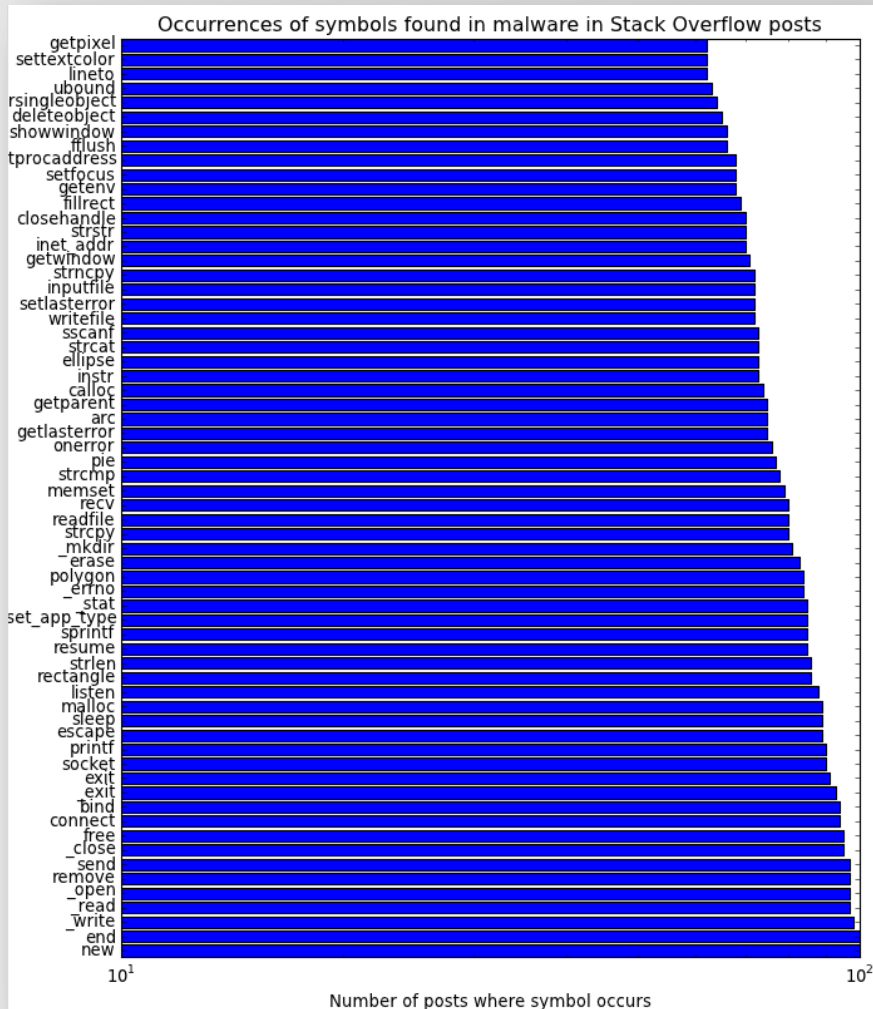
    // Get the window handle of calculator application.
    HWND hWnd = ::FindWindow( 0, _T( "Calculator" ));

    // Take screenshot.
    PrintWindow( hWnd,
                dc.GetSafeHdc(),
                0 );
}
```

We took 53 malware samples, unpacked them, and took the union of the function names appearing in their Import Address Tables.

Then we downloaded the entire body of Stack Overflow postings (6.5 million in all), loaded them into a database and indexed their text using a full text indexing system (SQLite3, to be precise).

Finally, we counted the number of posts in which each symbol appears.



Overall 77.6% of the function call names found in the malware appeared somewhere in the Stack Overflow posts. The mean number of posts for the function calls was 3195.78, with a standard deviation of 37034.2.

Punchline: the DLL functions called by a sample of malware binaries are discussed explicitly on the web

If we could mine these web documents, could we automatically say something about what the malware does?

Extracting useful information from the mapping: semantic networks for malware symbols

Hey all i want to login onto my works webpage with wininet, this is my current code:

```

2 int main()
  {
  HINTERNET hInet → InternetOpenA("UserAgent/1.0", INTERNET_OPE
  if(!hInet)
  {
    printf("hInet Failed!\n");
    return -1;
  }

  HINTERNET hConnection → InternetConnectA(hInet,"app.tamigo.c
  if (!hConnection)
  {
    InternetCloseHandle(hInet);
    printf("InternetConnectA failed!\n");
    return -1;
  }

  HINTERNET hRequest = HttpOpenRequestA(hConnection, "Accept",
  
```

Here InternetOpenA and InternetConnectA occur within 20 words of each other, so we add "1" to their co-occurrence count.

Next InternetCloseHandle and HttpOpenRequestA occur within 20 words of each other so we add "1" to their co-occurrence count as well.

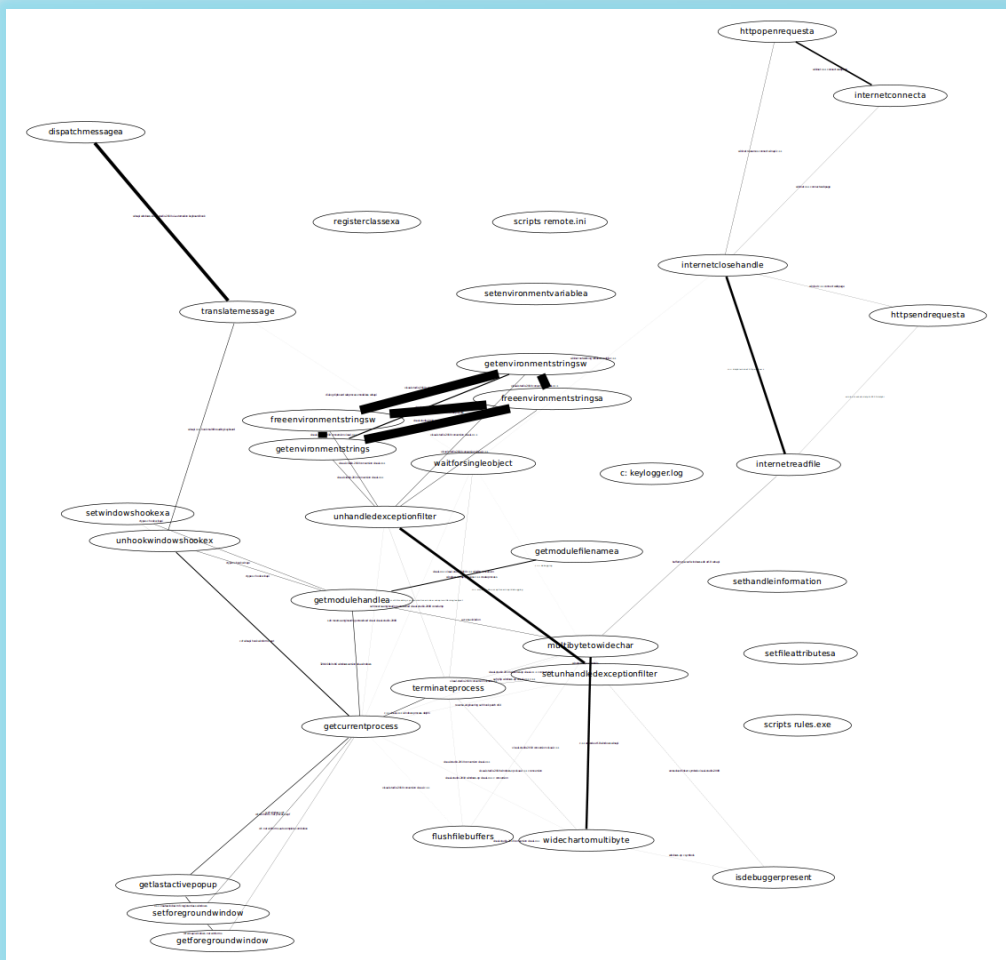
CREATING A SIMPLE SEMANTIC MAP OF THE MALWARE API

Our method is based on co-occurrence of a malware sample's function call names within 20-word windows within the StackOverflow posts.

By calculating overall call occurrence as well as pairwise co-occurrence relationships, we build up a network of co-occurrence probabilities. This statistical relationship strongly suggests functional and semantic dependence.

The edge weight between two imported function calls is computed by the following equation, which is equivalent to the minimum probability of "call A" appearing given the appearance of "call B" and vice versa:

$$\min\left(\frac{|Call_A \cap Call_B|}{|Call_A|}, \frac{|Call_A \cap Call_B|}{|Call_B|}\right)$$



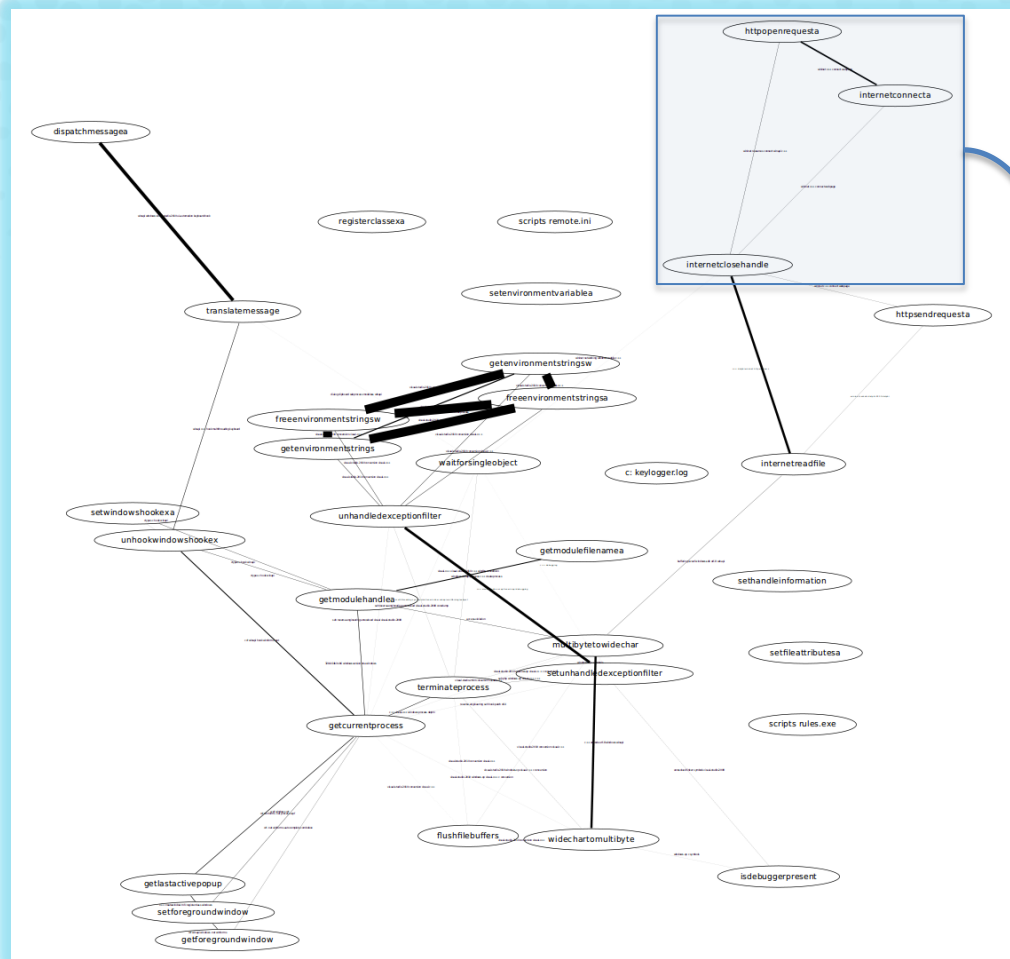
GRAPHICAL CLUSTER STRUCTURE

This example and most others exhibit a graph in which almost all nodes are mutually reachable

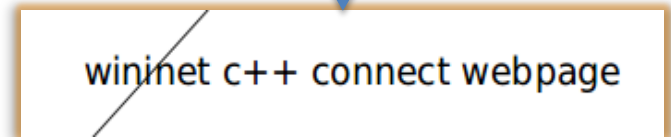
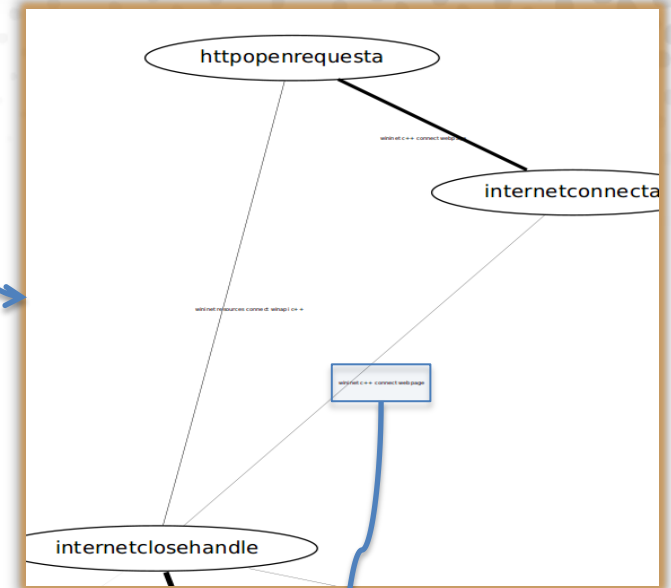
Graphical cluster structure aligns with intuitive sense of shared meaning and functional dependency between symbols

StackOverflow Based Semantic Network for One "Kbot" sample

Graph depicting Stack Overflow post co-occurrence relationships for strings in a single "Kbot" IRC bot sample



Zoomed in view: network component?



Zoomed in view: edge labeled with post tags

The next step, actionable intelligence: Explicit recognition of malware capabilities

- Basic idea:
 - We have a list of predefined capabilities (*takes screenshot, logs keystrokes*)
 - And a set of textual strings that we observe in a malware sample, such as file paths, registry keys and function names
 - We would like to know $P(\text{capability}|\text{symbol})$ for each capability given each symbol observed in the sample
- **Research problem: Can we somehow compute these probabilities by training on the StackOverflow corpus?**

Computing capability profiling model based on StackOverflow posts

- To compute $P(\text{capability}|\text{symbol})$

$$P(\text{capability}|\text{symbol}) = \frac{|posts_referencing_capability \cap posts_with_symbol|}{|posts_with_symbol|}$$

- To learn our model, pull out all symbols occurring in the malware corpus under analysis
- Compute $P(\text{capability}|\text{symbol})$ for every possible capability to symbol pair, caching them in a database as we go
- ***After this training phase, finding the probability of a capability given a symbol is a single constant time lookup of a mapping between symbol/capability and probability***
- Computing probability for a capability given a string of symbols can be performed as follows :

$$1 - \prod_i^N 1 - P(\text{capability}|\text{symbol}_i)$$

Probability that all symbol “sensors” are wrong

StackOverflow approach allows for minimal work in defining capabilities

Defining our capability patterns in a configuration file

```
patterns = {
  # format is capability name : StackOverflow query terms

  # -- general categories --
  "memory allocation":"tags:memory OR tags:heap OR title:memory",
  "string operations":"tags:string OR tags:unicode",
  "file operations":"title:file",
  "process operations":"title:process",
  "database operations":"tags:*sql*",
  "cryptography activity":"tags:*crypt",
  "shellcode related":"tags:shellcode OR title:shellcode",

  # -- network related --
  "network connectivity":"tags:socket OR title:socket OR title:tcp OR title:udp OR title:icmp",
  "network share activity": "tags:samba OR tags:SMB OR tags:CIFS",
  "web browser related activity": "tags:*browser*",
  "portscan activity": "tags:*portscan*",
  "SMTP transmission": "tags:SMTP OR tags:sendmail",
  "HTTP transmission": "tags:HTTP",
  "ICMP transmission":"title:icmp OR tags:icmp",
  "DNS transmission":"title:DNS OR tags:DNS",
  "irc activity":"tags:IRC title:IRC",

  # -- OS related --
  "system service activity": "tags:service OR tags:system-service",
  "authentication activity": "title:active AND title:directory",
  "privilege elevation": "title:elevation OR title:privilege OR tags:privilege OR tags:elevation",
  "thread injection": "title:thread AND title:injection",
  "anti-antivirus activity": "title:anti AND title:virus",
  "error handling": "tags:error AND tags:handling",
  "keylogging": "tags:*keylog* OR title:*keylog* OR title:keystroke OR tags:keystroke",
```

In contrast to rules based approaches, defining our patterns takes very little work

Because StackOverflow is a living corpus, our capability definitions will stay up to date with new APIs and programming trends

Preliminary empirical results indicate system accuracy is on par with expert rules based approaches but with vastly less work to create rules

Using the approach outlined above, our model “learns” what function calls are associated with what malware capabilities

```
[*] cryptography activity ---
[*] CryptUnprotectData 0.557620817844

[*] graphical user interface related ---
[*] GetPaletteEntries 0.921658986175
[*] GetNextDlgTabItem 0.921658986175
[*] ScrollDC 0.921658986175
[*] IsDlgButtonChecked 0.883218842002
[*] ValidateRect 0.875912408759
[*] StrokePath 0.854700854701
[*] CreatePalette 0.834492350487
[*] ExcludeClipRect 0.827966881325
[*] SetWorldTransform 0.824742268041
[*] RealGetWindowClass 0.815217391304
[*] InvalidateRgn 0.815217391304
[*] GetBkColor 0.796812749004
[*] PolyBezier 0.796812749004
[*] GradientFill 0.793231094659
[*] CloseFigure 0.786163522013
[*] CreatePatternBrush 0.779220779221
[*] CommDlgExtendedError 0.779220779221
[*] FillPath 0.774336283186
[*] MapWindowPoints 0.767918088737
[*] glPolygonOffset 0.746268656716
[*] glFogfv 0.746268656716
[*] SetMapMode 0.73589533933
[*] CreatePolygonRgn 0.730816077954
[*] InflateRect 0.725806451613
[*] IsDialogMessage 0.720164609053
[*] AdjustWindowRect 0.715990453461
[*] SetROP2 0.715990453461
[*] CombineRgn 0.701754385965
```

```
[*] command shell input ---
[*] SHChangeNotify 0.804073974806
[*] ShellExecute* 0.516467065868

[*] privilege elevation ---
[*] LookupPrivilegeValue* 0.528401585205
[*] SeDebugPrivilege 0.524475524476
[*] OpenProcessToken 0.407424173834

[*] upload activity ---
[*] InternetConnect* 0.585365853659
[*] HttpOpenRequest* 0.533997864009
[*] HttpSendRequest* 0.450788880541

[*] search for files ---
[*] SearchPath* 0.4212004212

[*] registry activity ---
[*] RegSetValue* 0.875912408759
[*] RegDeleteValue* 0.863060989643
[*] RegCloseKey 0.836186987338
[*] RegOpenKey* 0.818367810866
[*] RegQueryValue* 0.815217391304
[*] AdjustTokenPrivileges 0.486854917235
[*] LookupPrivilegeValue* 0.456621004566

[*] webcam spying ---
[*] capCreateCaptureWindow* 0.854700854701
```

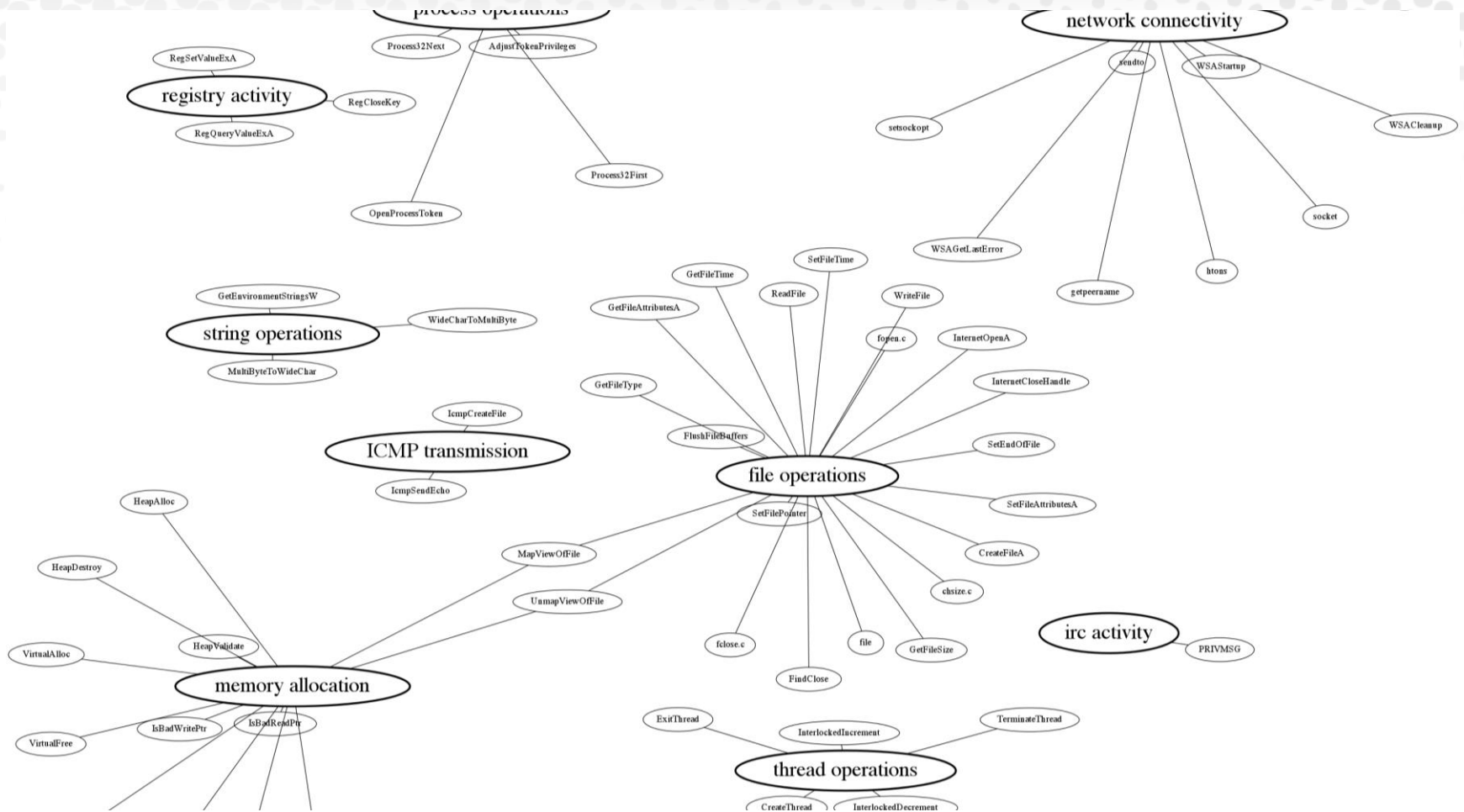
- By linking back to StackOverflow titles, tags and posts, model is also “self-documenting”
- In other words, the model can show why it “thinks” certain malware string symbols are associated with certain malware capabilities, by referencing the StackOverflow posts

Symbols found in the malware corpus

The probability of the compression/decompression capability given the symbol

Symbol	Probability	Example Post Titles
[*] compression / decompression activity ---		
[*] uncompress	0.748161568795	<ul style="list-style-type: none"> - Are there any downsides to using UPX to compress a Windows executable? - uncompress .Z file in C
[*] compress	0.714790586074	<ul style="list-style-type: none"> - How to compress JPEG images with ASP on Windows CE - Best way to compress HTML, CSS & JS with mod_deflate and mod_gzip disabled
[*] webcam spying ---		
[*] capCreateCaptureWindowA	0.871586287042	<ul style="list-style-type: none"> - Webcam usage in C# - Webcam Capture Resolution issues
[*] SetBitmapBits	0.79086115993	<ul style="list-style-type: none"> - Getting a snapshot from a webcam with Delphi - Getting a snapshot from a webcam with Delphi
[*] _setmode	0.477200424178	<ul style="list-style-type: none"> - Flash: BitmapData.draw(Video) ignores video height - Flash: BitmapData.draw(Video) ignores video height
[*] SendMessageA	0.42036431574	<ul style="list-style-type: none"> - C# Trying to use WM_CAP_SET_SEQUENCE_SETUP to allow me to maintain control of the application during video capture - C# Trying to use WM_CAP_SET_SEQUENCE_SETUP to allow me to maintain control of the application during video capture

Some example post titles in which both the symbol and the topic co-occur

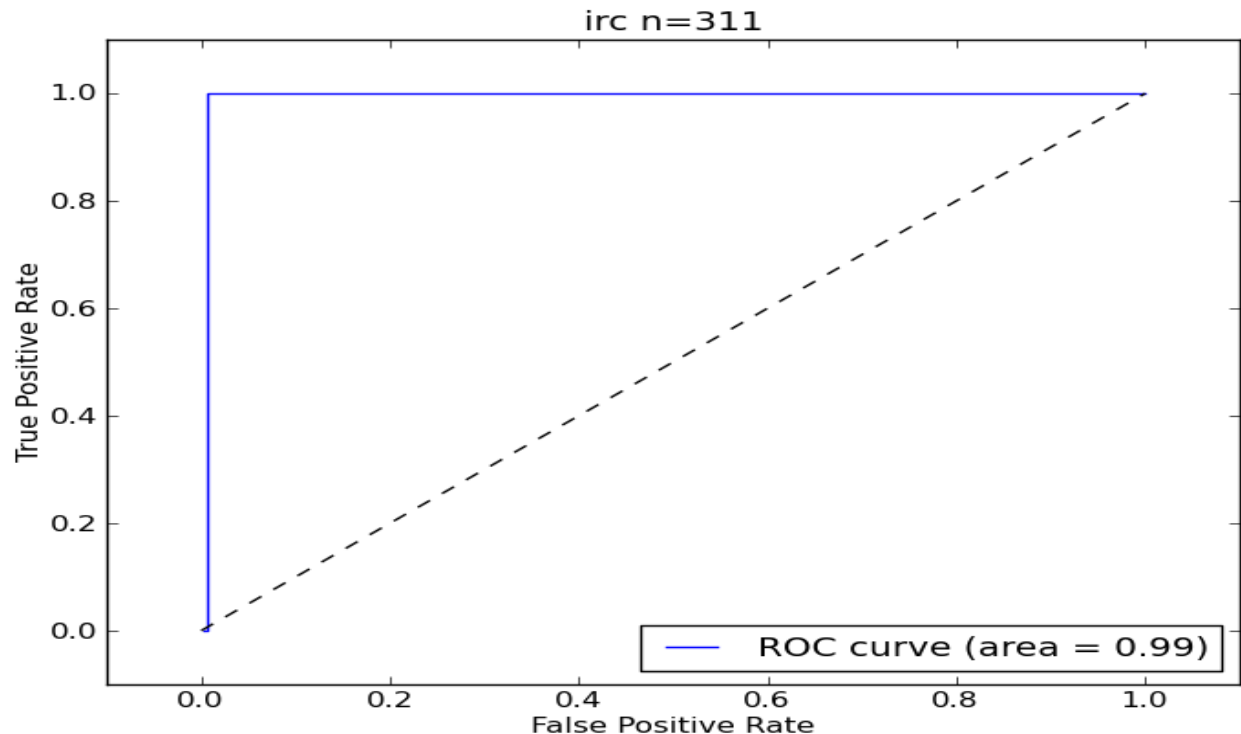


How accurate is all this?

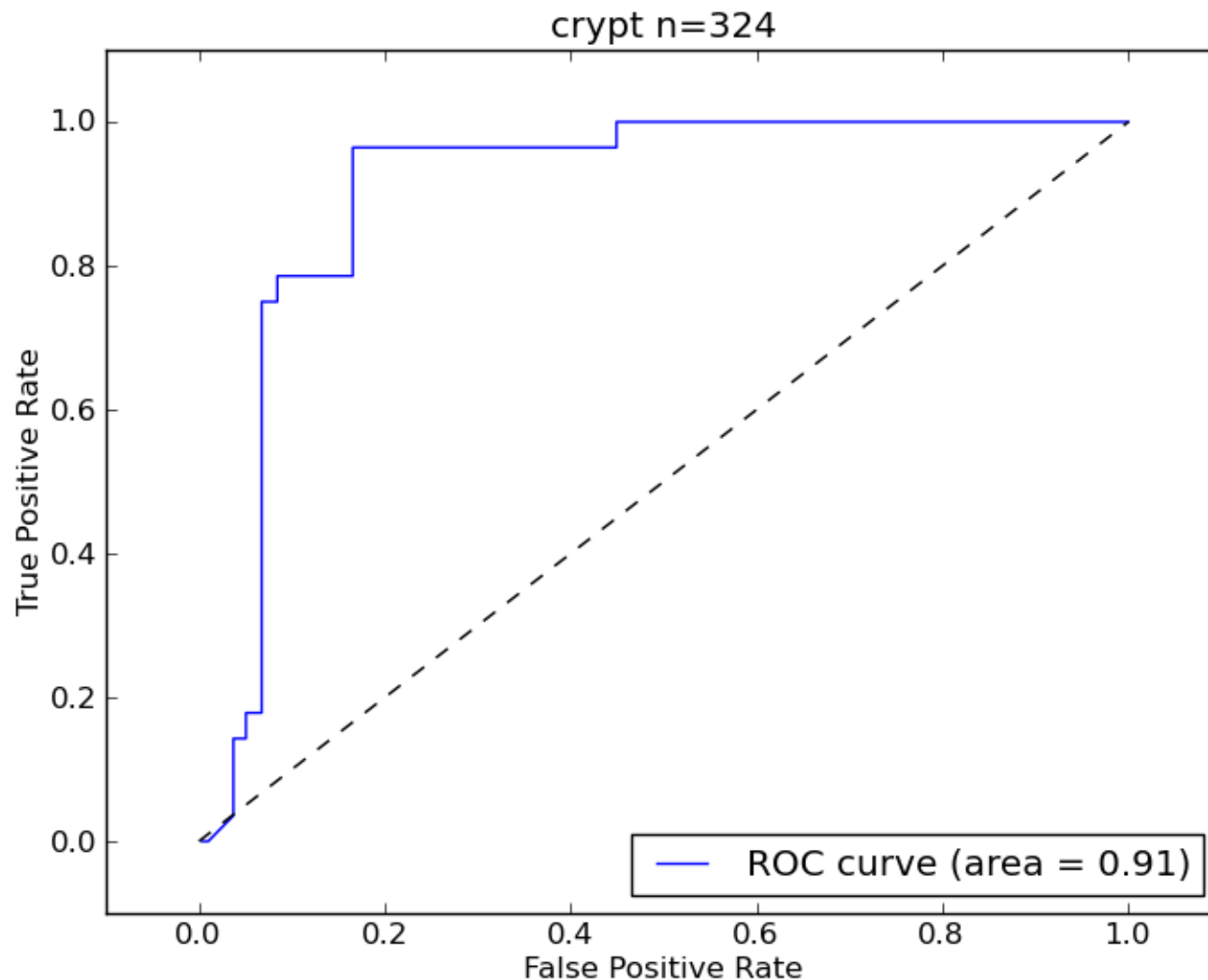
The answer in the form of ROC curves:

- Test dataset: ~300 Windows binaries, ~300 malware samples with known capabilities
- All samples came unpacked or we unpacked them ourselves
- We are assuming that an unpacking technology is deployed before running the CrowdSource approach...

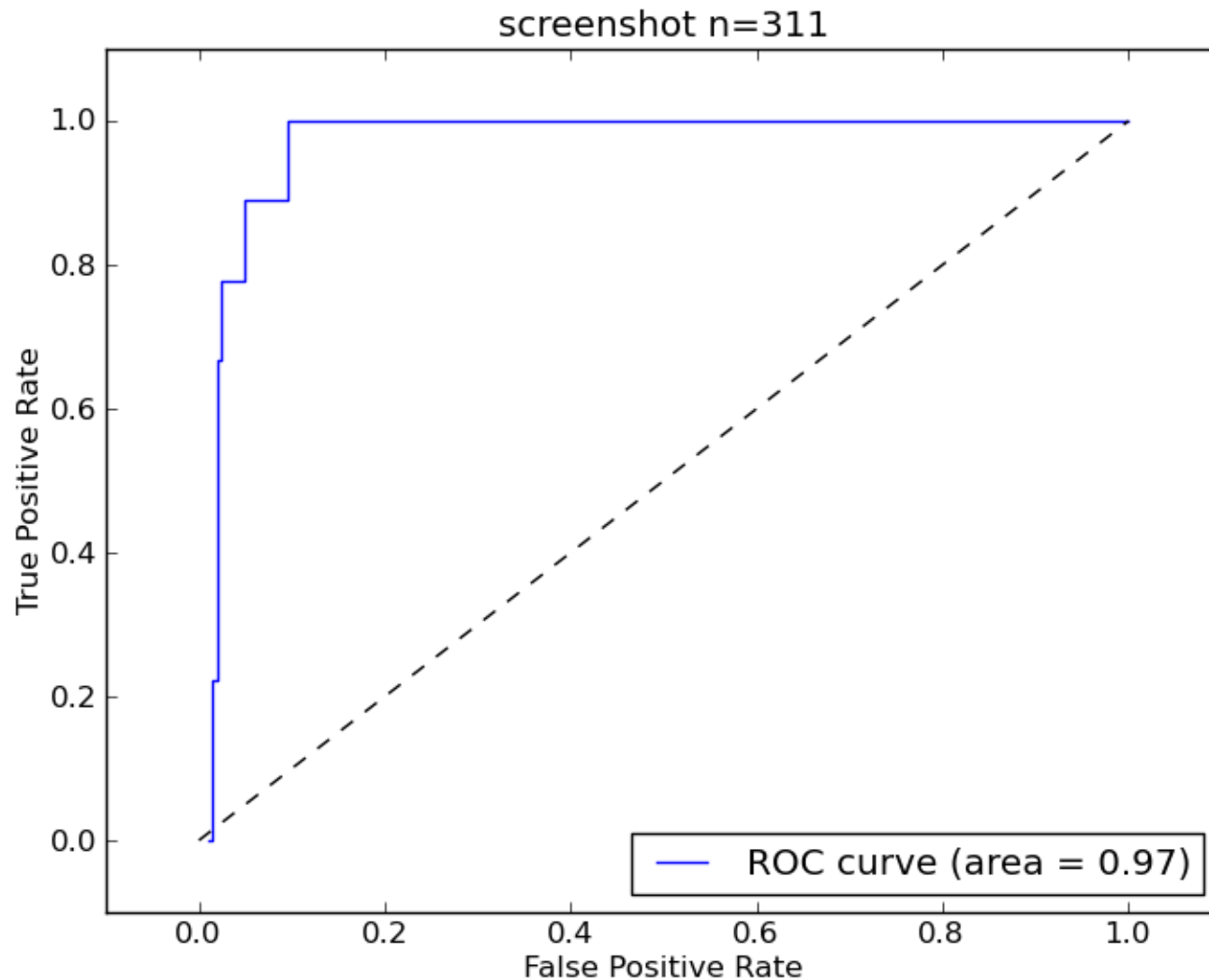
Perfect
results
detecting
IRC
capability



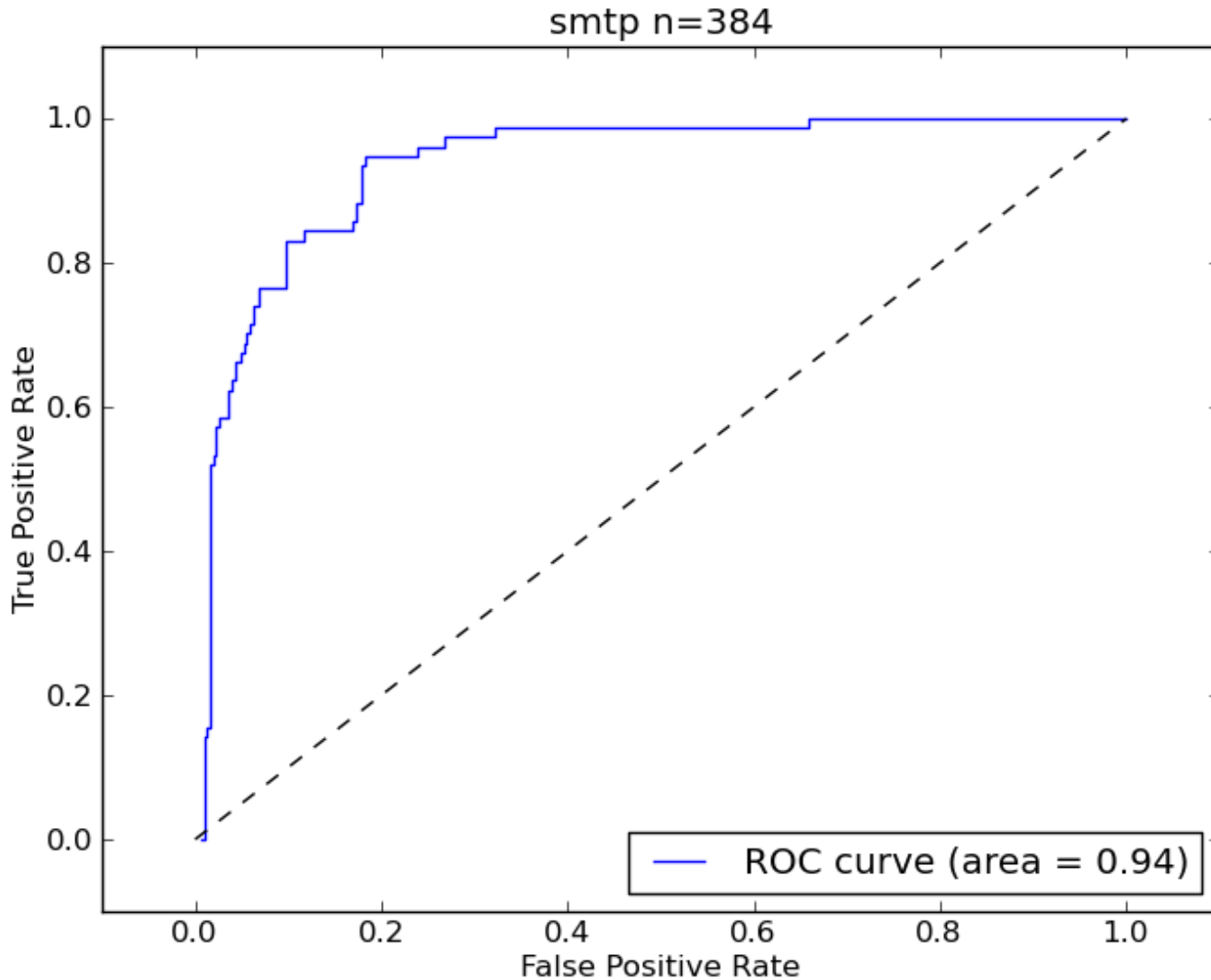
Detecting cryptography functionality in malware, decent performance on true and positive examples ..



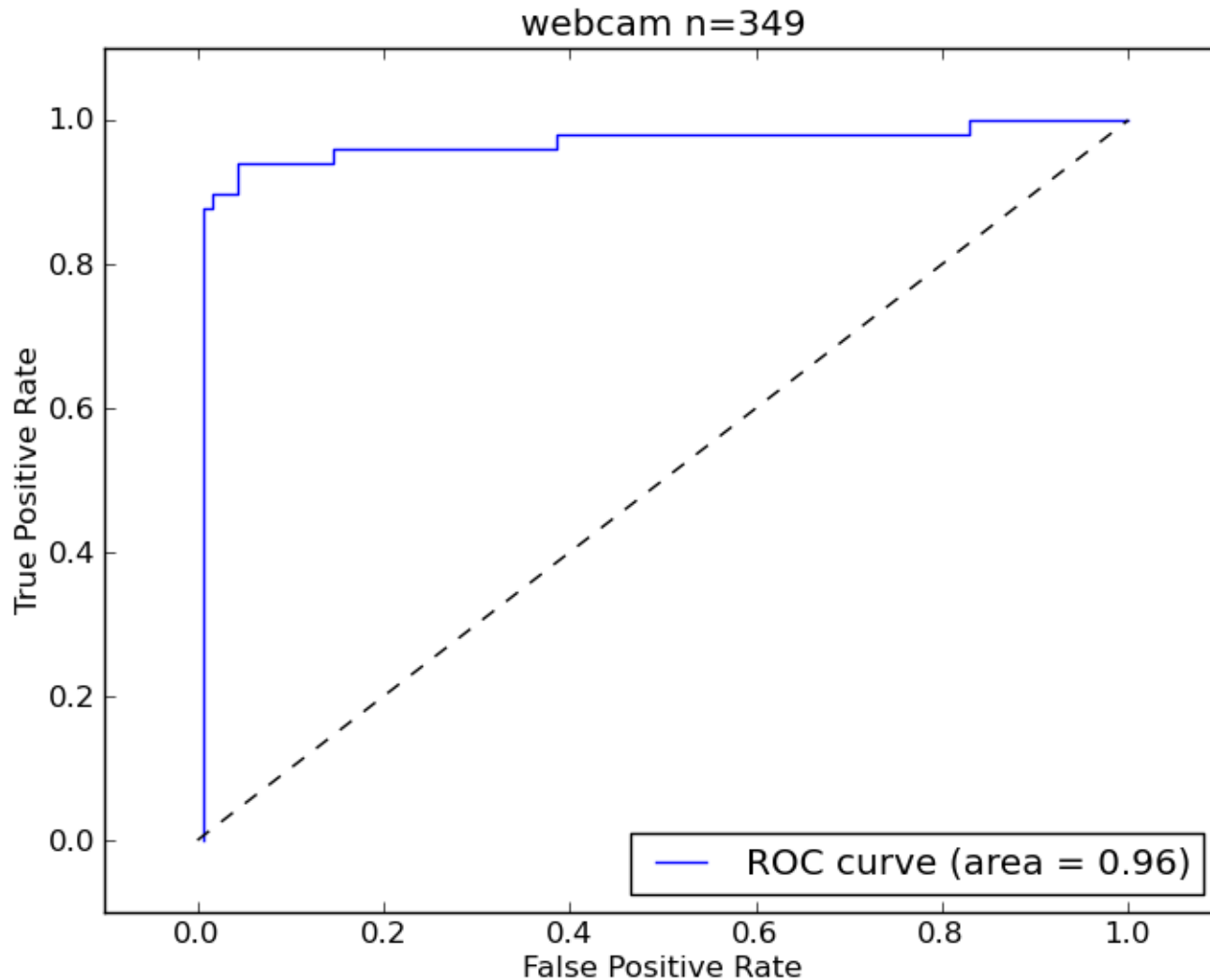
Detecting screenshot grabbing functionality...



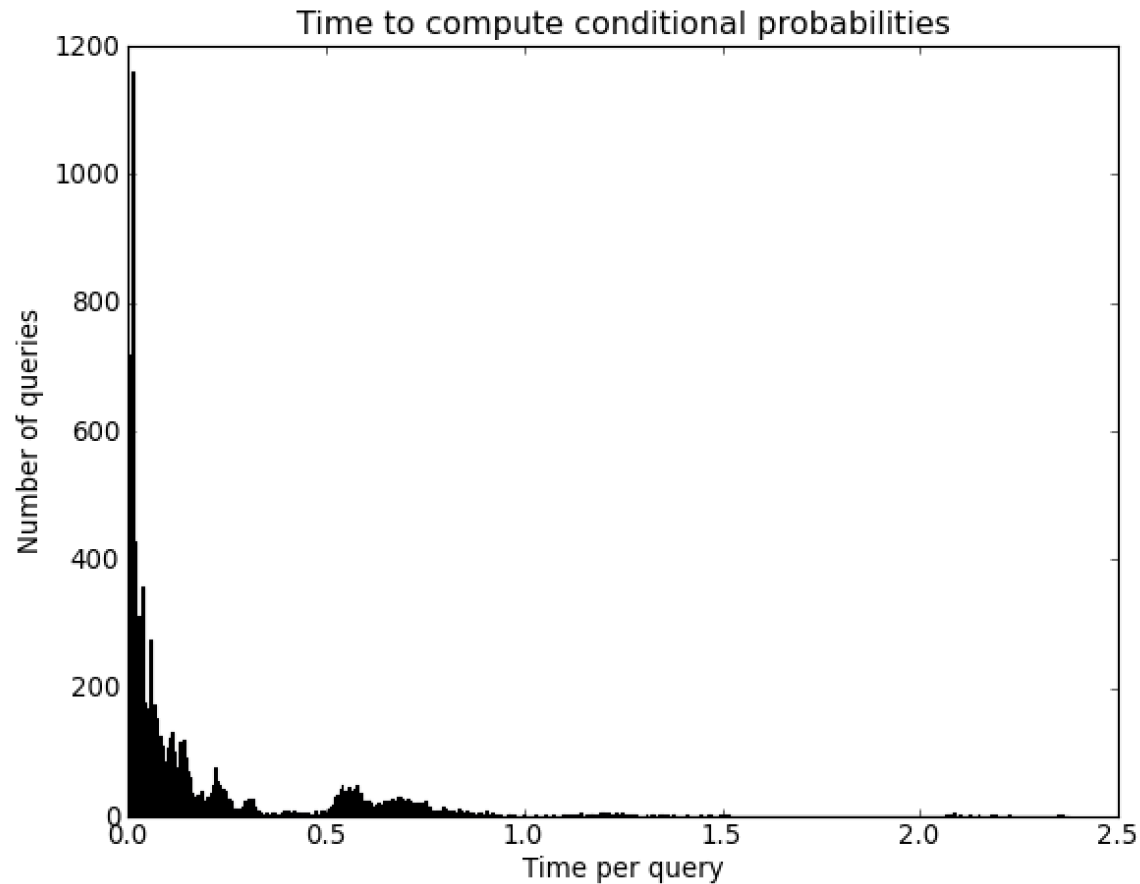
Detecting SMTP communication functionality, slightly less accurate ...



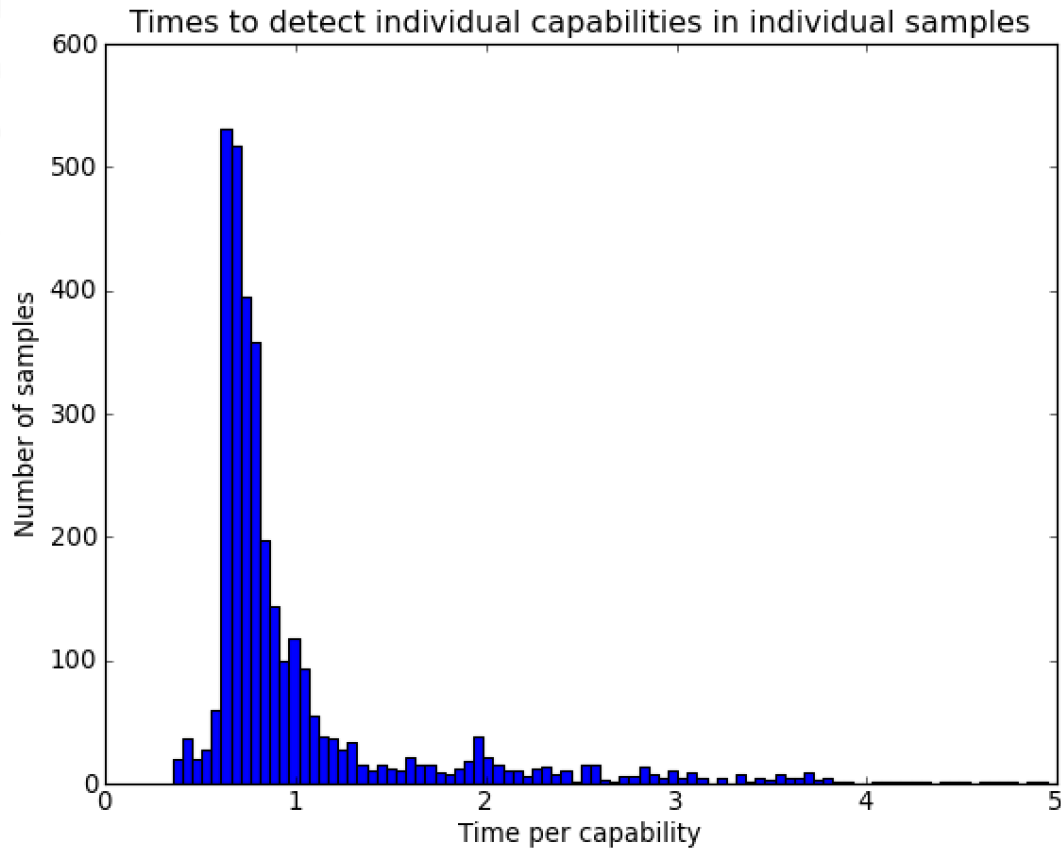
Detecting webcam functionality, quite accurate ...



Speed test: Speed of database queries for retrieving relevant posts



Speed to run capability detection on a sample assuming cached queries:



- In November, we will release an open source version of CrowdSource to run on Debian based Linux systems
- We will continue to develop our statistical model to extract more information from the technical documents
- As our approach grows in accuracy we plan to explore detecting more malware capabilities