Will Ratner
Michael Patterson

Automated Malware Feed False Positives: Citadel Malware

For our final project, we looked at automated malware feeds pertaining to the Citadel Trojan. The Citadel Trojan is an offspring of the popular Zeus crime kit whose main goal is to steal banking credentials by capturing keystrokes and taking screenshots or videos of the victims' computers.

Our goal for this project was to examine these feeds which were separated into three separate categories. These include Antivirus information, Yara information, and Snort Information. All of these categories typically include the Trojan file, and the analyzed results of this file from VirusTotal.com.

Doing some initial online research, we found that one dead giveaway to a Trojan being the Citadel Trojan is if it contains the string "Coded by BRIAN KREBS for personal use only. I love my job & wife." Brian Krebs is the name of a security researcher who researches commercial bots such as Zeus, SpyEye and Citadel. So here is one rule that gives a really good way to detect the Citadel Trojan; however, what about the rules that are not as definitive? This is what we aimed to figure out, that is, what rules being used may not be as helpful in detecting the extremely malicious Citadel Trojan.

The difficult part of this project was figuring out just what rules were causing these binaries in the feeds to be flagged. The feeds contained no set of rules that were used to determine that these binaries were malicious. The antivirus report gave us regular expression matches, but that didn't really give us any indication of a false positives. Most of the antivirus reports we looked at by hand had similar matches. An example can be seen below:

{"AVG":"Crypt.BXNK","AhnLab":"Trojan/Win32.Zbot","Avast":"Win32:MalPack-G","Av
{"ira":"TR/PSW.Zbot.2295","BitDefender":"Gen:Variant.Kazy.172829","Comodo":"Un
{"classifiedMalware","DrWeb":"Trojan.PWS.Panda.2401","Emsisoft":"Gen:Variant.K
{"azy.172829","Emsisoft":"Win32.Worm.Brontok.G","Eset":"Win32/Kryptik.BALJ","E
{"set":"Win32/Kryptik.BAMB","FSecure":"Gen:Variant.Kazy.172827","FSecure":"Gen
{":Variant.Kazy.172829","Fortinet":"W32/Kryptik.AGAJ!tr","GData":"Gen:Variant.
{"Kazy.172829","Ikarus":"Trojan.Crypt","K7":"EmailWorm (0040f4131
{")","Kaspersky":"Trojan-Spy.Win32.Zbot.ljus","McAfee":"PWS-Zbot-FAZB!00ACE508
{"51C1","Microsoft":"PWS:Win32/Zbot","Norman":"win32/Dorkbot.GUU","Sophos":"Ma
{"l/EncPk-AKC","Sophos":"Sus/UnkPack-C","Sunbelt":"Trojan.Win32.Kryptik.bamb",
{""Symantec":"Backdoor.Trojan","VirusBlokAda":"BScope.Trojan.MTA.0661","VirusB
{"uster":"TrojanSpy.Zbot!Y3OzppNCu8w","VirusBuster":"Win32.Ipamor.B"}

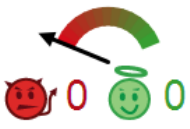
These results come from virustotal.com, so we decided to go check out that website and see what other information we could find about these binaries. We found out that virus total gives us the total number of antiviruses used to scan the binary submitted, and the number of positive hits that the antiviruses scanned. We also noticed a pattern that the higher the detection rate, the more malicious the file was considered. It gave us a general summary seen below:

SHA256: a7fe6136f148291ca0b6bf0586394972144826d9d43f39170e7c16976df82ab0

File name: soft.exe

Detection ratio: 46 / 51

Analysis date: 2014-04-17 00:16:27 UTC (2 weeks, 6 days ago)



Based on these findings, we felt a good way to filter through the hundreds if not thousands of files we had was to figure out which ones had a low detection rate. In order to do this, we used the virus total API, and an API client found on Github (https://github.com/Gawen/virustotal) to streamline the process even more.

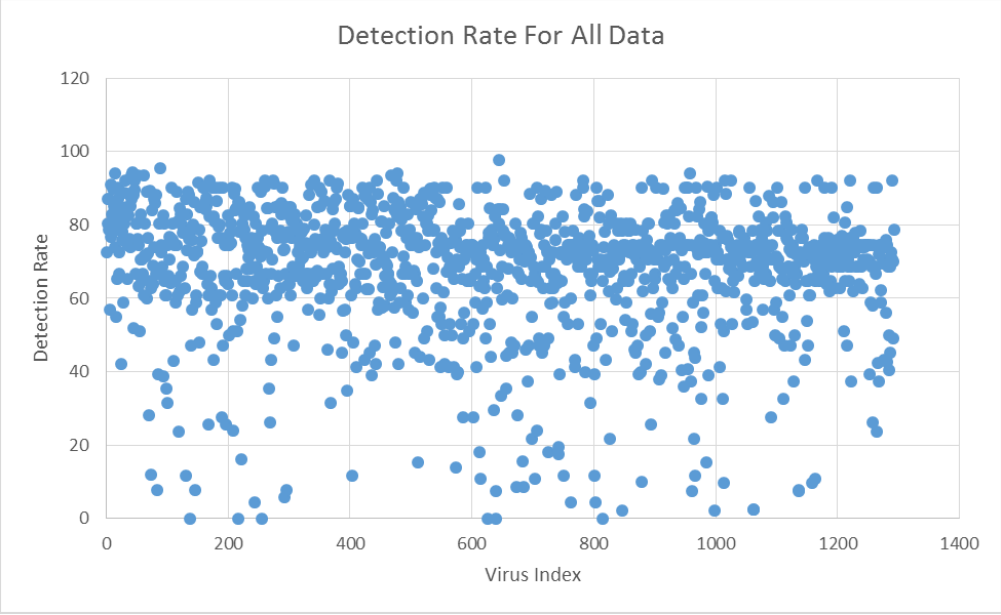
Our idea was to find the average number of positive detections for all of the binaries and find what files had abnormally low detection rates and examine them more closely. We wrote a python script that would print the MD5 hash of the binary, the number of positive detections, and the total number of antiviruses used. We would use that to compute the average and standard deviation of all the positive detections and total tests run. We would then find which binaries had a positive detection rate two standard deviations below the average. This would signify unusual behavior, such as the possibility of a false positive.

Limiting ourselves to more promising files will give us the ability to examine and analyze those more closely and efficiently, especially pertaining to their other rules sets in Snort and Yara.

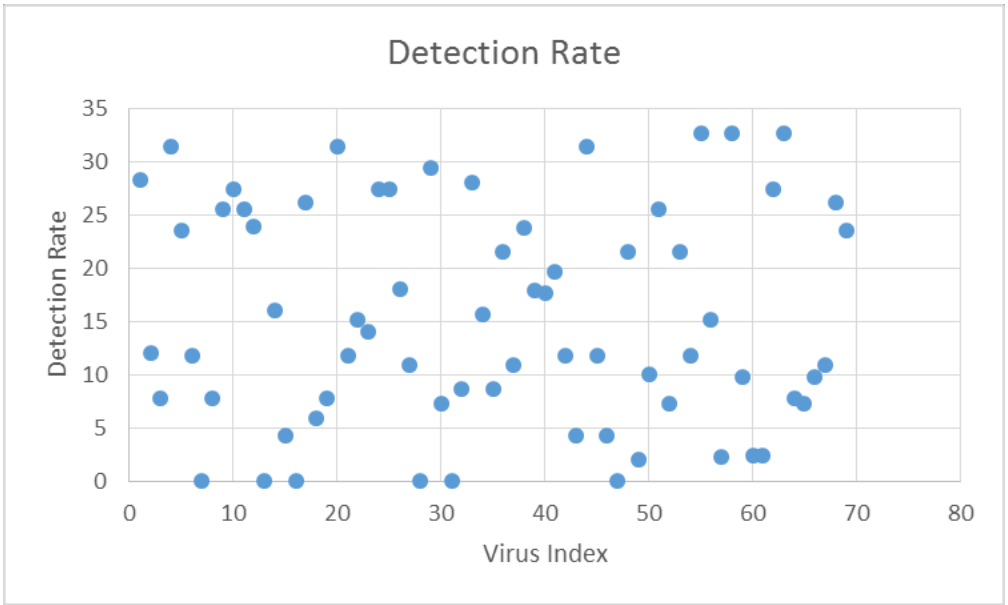
We used the python script with the virus total API to first gather all of our required data and place it in a file. We collected data from approximately 650 binaries each. It took about five or more hours each to collect this data as the API would only allow us to query the server four times a minute. We created a file with about 1300 data entries that looked like this:

bc83f34e3faea7313e157991a1143982 37 51

The first column is the MD5 hash of the binary, the second is the number of positive virus detections, and the third is the total number of scanners used. Some binaries did not have reports which we assume is because they were only detected by either Snort or Yara. We would come back later to analyze these. The next thing we did was get the detection rate for each binary. This was done by just dividing the number of positive detections by the total number of scanners used. This would give us the percentage or detection rate of the virus for each binary. We wrote another python script to import this data into an excel spreadsheet and plot it. The plot can be seen below:

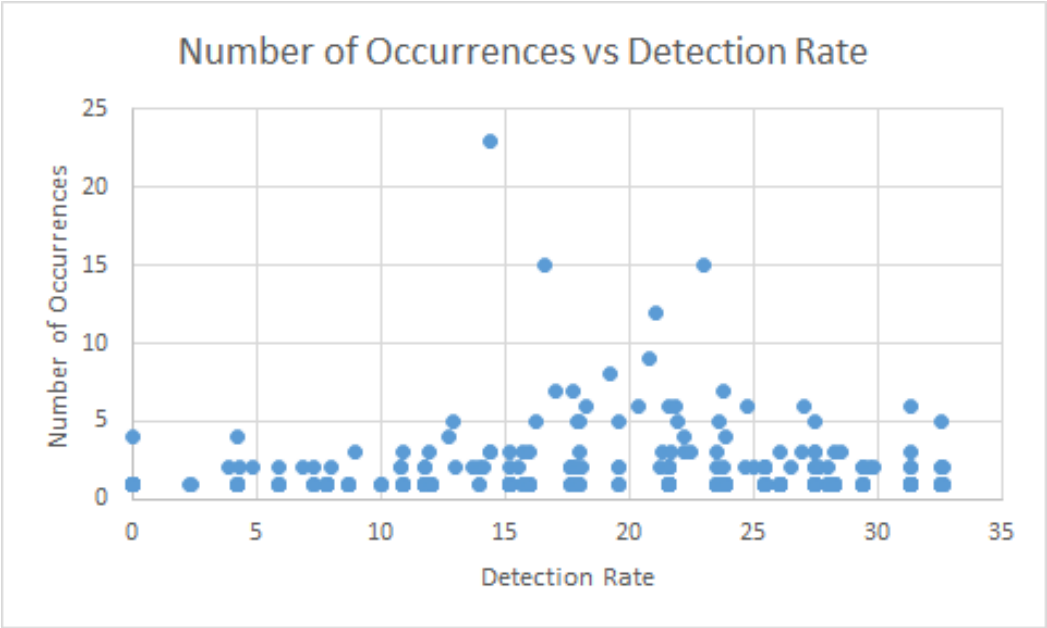


As you can see from the chart, the majority of the virus binaries have a detection rate between 60 and 90 percent. However, you can also see that there are quite a few outliers that are below what appears to be the average. Our python script that computed the detection rates also gave us an average detection rate as well as the standard deviation. The average turned out to be 68.0324795695 while the standard deviation was 17.556042963. Knowing this, and what we know from basic statistics, any value that is outside of two standard deviations from the average is considered unusual. So in order to obtain a more manageable number of possible false positives to examine, we created a document only containing the binaries that have detection rates two standard deviations below the average. There were approximately 69 values with this attribute and you can see them plotted below:



You will notice that some of these values have a detection rate of 0. This tells us that these binaries were probably not detected by antivirus software but instead by either snort or yara rules. These are particularly interesting because you would expect them to be picked up by both snort and yara as well as virus scanners based on the large number that were detected in the first graph.

In order to refine our search even more in regards to the antivirus results, we looked at which rules these predicted false positives were using to flag down the binaries as the Citadel Trojan. The majority of the binaries from the feed contained antivirus.json files (as seen on the first page) which tells us exactly which rule was triggered by each anti virus software. With this, we decided that a good way to find out which rules may be causing these possible false positives is to get a count of the number of times a rule was triggered among the binaries that could possibly be false positives and associate them with their respected detection rate. Doing this would will help us make a more accurate estimation about which binaries are possible false positives. Using the same script to import data into an excel spreadsheet, we used excel to create a graph to visualize our data as on the next page. The text file containing all this data is included with the project files and is called “false-positives-av-rule-report.txt.



Although there is quite a bit of overlap in this chart, the intended result is still the same. We can see that there are a few instance where there have been a noteworthy amount of occurrences of a specific rule at a relatively low detection rate. Since there are too many data points to put the actual names of the rules being, here are some the rules that had the highest occurrences:

Detection Rate	Occurrences	Rule
14.40318641	23	Win32/Spy.Zbot.AAO
16.58129276	15	Suspicious.Insight
22.96388555	15	Trojan.Win32.Generic!BT
21.11138045	12	Win32:Malware-gen
20.82120767	9	Trojan.PWS.Panda.2401
19.25085251	8	Virus.Win32.Heur.p
17.07970885	7	HEUR:Trojan.Win32.Generic
17.72835091	7	UnclassifiedMalware
23.76470588	7	Trojan
18.27813148	6	WS.Reputation.1
20.41291232	6	PWS:Win32/Zbot
21.56862745	6	Trojan.GenericKD.1659781
21.89542484	6	PE:Malware.XPACK-HIE/Heur!1.9C48
24.73083876	6	Spyware
27.0173454	6	Mal/Generic-S
31.37254902	6	Trojan.GenericKD.1649427

One interesting point from this information is that the most common occurrence occurs with the lowest detection rate. The “Win32/Spy.Zbot.AAO” This tells us that there is a high chance that this rule may be causing false positives. It is definitely a binary that we will want to look closer at as well as the rule. Although this gives yet another good estimation, we can not be 100% sure. We need to see if these rules were also used in binaries that we have deemed to have no chance of being false positives. To do

this, we wrote yet another python script to analyze both files that contain possible false positives and no chance of being false positives. The file has the following format:

<rule> <# FP occurrences>,<avg FP occurrence detection rate>,<# NFP occurrences>,<avg NFP occurrence detection rate>,<% of the time this rule causes a false positive>

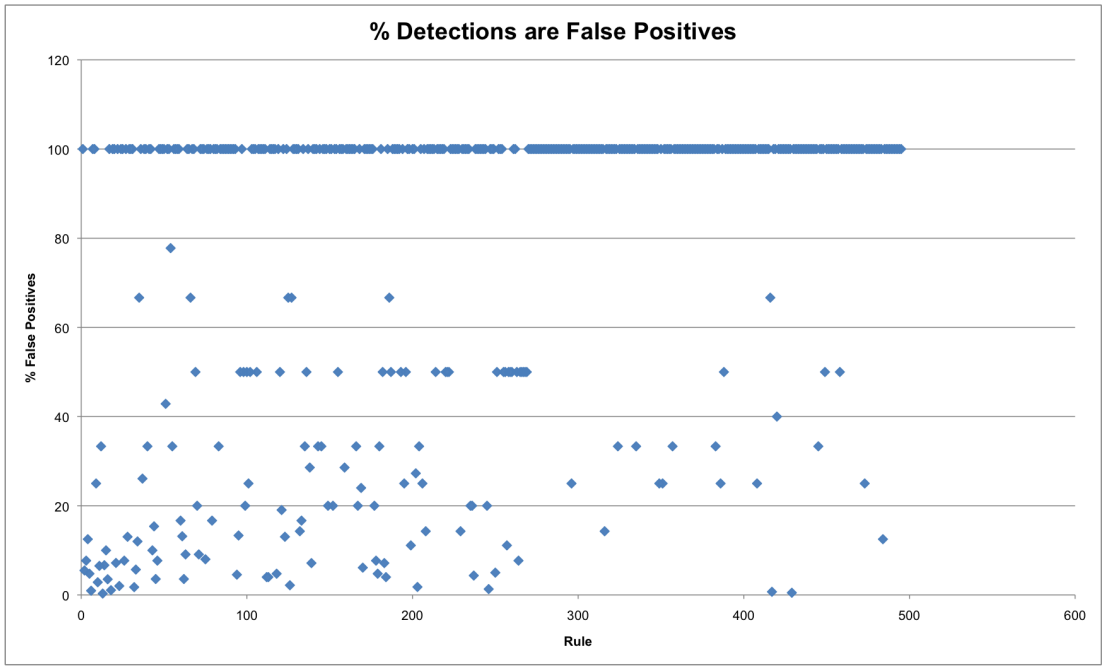
FP = false-positive

NFP = non-false-positive

avg NFP occurrence detection rate is -1 if the # NFP occurrences is 0 which can cause the last field to be 100.0

The rules with the higher percentages in the last field are the ones that cause false positives most often, and are the least useful. Out of those with high percentages, the ones, with an average NFP occurrence detection rates farthest from the average are the worst. The rules with 100% FP rates are the rules that we could recommend getting rid of. Using this rule, we can get a more precise estimation of false positives. We wrote all of these to a values to the file "av-rule-ratings.txt".

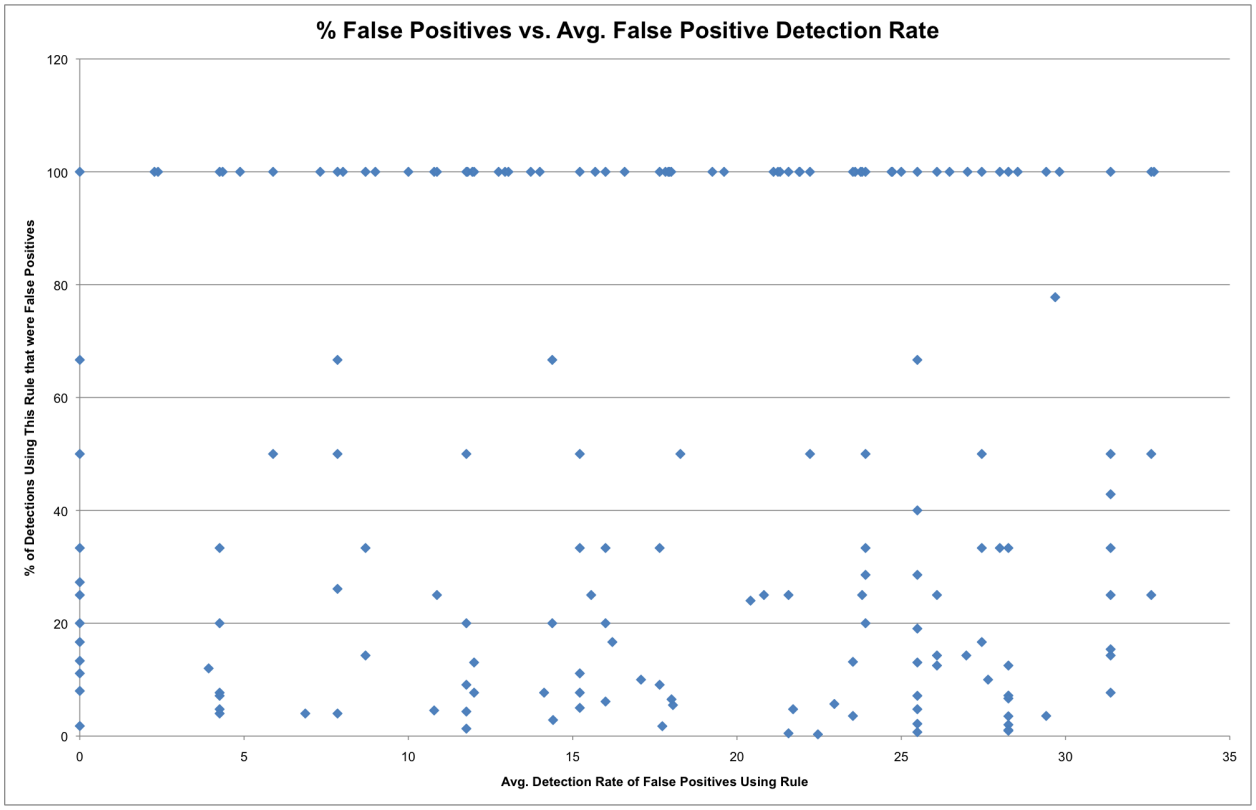
A graph showing the percentage of time each rule was used to produce a false positive is shown below:



Analyzing this data, we discovered that the rule that we thought was surely causing false positives "Win32/Spy.Zbot.AAO" was not as rare as we thought. It turns out that that rule only causes a possible false positive 6.51558073654 percent of the time. This definitely tells us that we can throw this rule out as being a culprit to causing false positives.

We did, however, find many instances where there was a number of high false positive occurrences and a low number of non-false positives. This gives us a good indication that these rules have a high probability of causing false positives.

The following graph shows the percentage of time a rule was used to detect a false positive vs. the average detection rate of the false positives it detected. (The rules with an average detection rate of 0 were detected by either Snort or Yara.)



Although there were many rules that caused false positives 100% of the time (caused no true positives), we are most interested in the ones which either occurred the most (had a high number of false-positive occurrences) or also had low detection rates (low avg. false-positive occurrence detection rates).

The rules that really stuck out were:

Rule	# FP occurrences	Avg FP occurrence detection rate	# NFP occurrences	Avg NFP occurrence detection rate	% of the time this rule causes a false positive
Virus.Win32.Heur.p	8	19.25085251	0	-1	100
Trojan.GenerickDV.1054267	4	4.255319149	0	-1	100
Suspicious File	1	2.272727273	0	-1	100
Packed.Win32.Katusha.o	1	2.380952381	0	-1	100

Now that we have a few of the most likely false positives, we will examine the snort, yara, and IOC rules to see if anything in those rules may be linked to these possible false positive binaries.

The Yara rule for Citadel is:

```
rule crime_win_citadel_mem_dev
{
  meta:
      description = "Rule to find unpacked zbot/citadel strings in memory"
      author = "@BryanNolen"
      yaraexchange = "No distribution without author's consent"
      date = "2013-07"
      md5 = "1bac48fed97d45df6c80d076bc5d6b7f f26c836c5a06dff93abee029da21bb89 8d9568a83bd9e85cabd086b368cff020"
    strings:
      $batch_a = "if exist \"%s\" goto d" ascii
      $batch_b = "del /F \"%s\"" ascii
      $user_agent_function = "ObtainUserAgentString" ascii
      $user_agent_dll = "urlmon.dll" ascii

      $optional_mutex_a = "Global\\" ascii
      $optional_mutex_b = "Local\\" ascii
      $optional_priv_function = "SeTcbPrivilege" wide ascii

      $optional_url_mask = "http://%02x%02x%02x%02x%02x%02x%02x%02x.com/%02x%02x%02x%02x/%02x%02x%02x%02x.php" wide ascii
      $optional_bot_string_a = "%BOTID%" wide ascii
      $optional_bot_string_b = "%BOTNET%" wide ascii
      $optional_mask = "s%08X%08X%08X%08X" wide ascii
    condition:
      ($batch_a and $batch_b and $user_agent_function and $user_agent_dll) and any of ($optional*)
}
```

While the Snort rules are:

```
#
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET TROJAN Zeus/Citadel Control Panel Access (
Outbound)"; flow:established,to_server; content:".php?m=login"; fast_pattern:only; http_uri; nocase;
content:"user="; http_client_body; depth:5; nocase; content:"pass="; http_client_body; nocase;
reference:url,xylithreats.free.fr/public/; reference:url,www.xylibox.com/2012/10/citadel-1351-rain-edition.
html; classtype:trojan-activity; sid:2015825; rev:7;)
#
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"ET TROJAN Zeus/Citadel Control Panel Access (
Inbound)"; flow:established,to_server; content:".php?m=login"; http_uri; fast_pattern:only; nocase;
content:"user="; depth:5; http_client_body; nocase; content:"pass="; http_client_body; nocase;
reference:url,xylithreats.free.fr/public/; reference:url,www.xylibox.com/2012/10/citadel-1351-rain-edition.
html; classtype:trojan-activity; sid:2015826; rev:6;)
#
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET TROJAN Citadel API Access Iframer Controller (
Outbound)"; flow:established,to_server; content:"/api.php/"; http_uri; fast_pattern:only;
content:"/iframer/"; http_uri; nocase; reference:url,xylithreats.free.fr/public/; reference:url,www.xylibox.
com/2012/10/citadel-1351-rain-edition.html; classtype:trojan-activity; sid:2015827; rev:5;)
#
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"ET TROJAN Citadel API Access IFramer Controller (
Inbound)"; flow:established,to_server; content:"/api.php/"; http_uri; fast_pattern:only; content:"/iframer/";
http_uri; nocase; reference:url,xylithreats.free.fr/public/; reference:url,www.xylibox.com/2012/10/citadel-
1351-rain-edition.html; classtype:trojan-activity; sid:2015828; rev:6;)
#
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET TROJAN Citadel API Access VNC Controller (
Outbound)"; flow:established,to_server; content:"/api.php/"; http_uri; fast_pattern:only; content:"/vnc/";
http_uri; nocase; content:"botI"; http_uri; nocase; reference:url,xylithreats.free.fr/public/;
reference:url,www.xylibox.com/2012/10/citadel-1351-rain-edition.html; classtype:trojan-activity; sid:2015829;
rev:5;)
#
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"ET TROJAN Citadel API Access VNC Controller (
Inbound)"; flow:established,to_server; content:"/api.php/"; http_uri; fast_pattern:only; content:"/vnc/";
http_uri; nocase; content:"botI"; http_uri; nocase; reference:url,xylithreats.free.fr/public/;
reference:url,www.xylibox.com/2012/10/citadel-1351-rain-edition.html; classtype:trojan-activity; sid:2015830;
rev:5;)
#
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET TROJAN Citadel API Access Bot Controller (
Outbound)"; flow:established,to_server; content:"/api.php/"; fast_pattern:only; http_uri; content:"/bots/";
http_uri; nocase; content:"botI"; http_uri; nocase; reference:url,xylithreats.free.fr/public/;
reference:url,www.xylibox.com/2012/10/citadel-1351-rain-edition.html; classtype:trojan-activity; sid:2015831;
rev:5;)
#
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"ET TROJAN Citadel API Access Bot Controller (
Inbound)"; flow:established,to_server; content:"/api.php/"; fast_pattern:only; http_uri; content:"/bots/";
http_uri; nocase; content:"botI"; http_uri; nocase; reference:url,xylithreats.free.fr/public/;
reference:url,www.xylibox.com/2012/10/citadel-1351-rain-edition.html; classtype:trojan-activity; sid:2015832;
rev:5;)
#
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"ET TROJAN Citadel API Access Video Controller (
Outbound)"; flow:established,to_server; content:"/api.php/"; http_uri; fast_pattern:only; content:"/video/";
http_uri; nocase; content:"botI"; http_uri; nocase; reference:url,xylithreats.free.fr/public/;
reference:url,www.xylibox.com/2012/10/citadel-1351-rain-edition.html; classtype:trojan-activity; sid:2015833;
rev:5;)
#
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"ET TROJAN Citadel API Access Video Controller (
Inbound)"; flow:established,to_server; content:"/api.php/"; fast_pattern:only; http_uri; content:"/video/";
http_uri; nocase; content:"botI"; http_uri; nocase; reference:url,xylithreats.free.fr/public/;
reference:url,www.xylibox.com/2012/10/citadel-1351-rain-edition.html; classtype:trojan-activity; sid:2015834;
rev:6;)
```

and finally the IOC rules are:

```
<?xml version="1.0" encoding="us-ascii"?>

<ioc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" id="226666e7-e748-4488-8d86-a641614a4cfa"
last-modified="2013-03-12T17:59:57" xmlns="http://schemas.mandiant.com/2010/ioc">

  <short_description>Citadel</short_description>
  <description>http://www.mcafee.com/us/resources/white-papers/wp-citadel-trojan.pdf
http://blog.malwarebytes.org/intelligence/2012/11/citadel-a-cyber-criminals-ultimate-weapon/
http://www.threatexpert.com/report.aspx?md5=91b64502a89d6c47d1adbde3ebbf2532</description>
  <authored_by>Megan</authored_by>
  <authored_date>2013-03-12T16:15:00</authored_date>
  <links />
  <definition>
        <Indicator operator="OR" id="28123200-9893-4bbf-8b8b-c202c618be64">
        <IndicatorItem id="f130f7fe-8d1d-498b-a9d8-47cec37d3922" condition="contains">
        <Context document="Network" search="Network/DNS" type="mir" />
        <Content type="string">msecure.su</Content>
        </IndicatorItem>
        <IndicatorItem id="1b619eb3-4c35-4c47-bc82-10962d5a4771" condition="is">
        <Context document="PortItem" search="PortItem/remoteIP" type="mir" />
        <Content type="IP">198.162.116.16</Content>
        </IndicatorItem>
        <IndicatorItem id="219264d9-3355-4da8-81d6-10bfe8ae23de" condition="is">
        <Context document="PortItem" search="PortItem/remoteIP" type="mir" />
        <Content type="IP">93.186.171.133</Content>
        </IndicatorItem>
        <IndicatorItem id="2124ea6c-a4b5-45d4-af46-3e42eec0901b" condition="contains">
        <Context document="Network" search="Network/DNS" type="mir" />
        <Content type="string">proscitomash.com</Content>
        </IndicatorItem>
        <IndicatorItem id="8e31fb06-9bf6-4d62-92b0-adb9daff567a" condition="contains">
        <Context document="Network" search="Network/DNS" type="mir" />
        <Content type="string">quliner.ru</Content>
        </IndicatorItem>
        <IndicatorItem id="40862803-fb91-492d-8f7f-136c7d5ebe49" condition="is">
        <Context document="FileItem" search="FileItem/Md5sum" type="mir" />
        <Content type="md5">91B64502A89D6C47D1ADBDE3EBBF2532</Content>
        </IndicatorItem>
        <IndicatorItem id="76533f85-34a2-4812-8311-9e96c07c913c" condition="is">
        <Context document="PortItem" search="PortItem/remoteIP" type="mir" />
        <Content type="IP">209.85.229.104</Content>
        </IndicatorItem>
        </Indicator>
  </definition>
</ioc>
```

With all of these rules, we can try to analyze what other rules besides the antivirus ones may be causing false positives. This will be some what difficult as in nearly all cases since we do not know exactly what rules of snort, yara, and ioc were set off to cause these binaries to be flagged. There are also a relatively small amount of rules which lowers the probability of any of these three categories containing rules that may result in false positives.

Starting with the Yara rules that we found, it is difficult to see exactly what specific rules are being used to scan the binaries. It appears to be more of a template than anything else. It is checking if various strings are present. These strings could be similar to that of the strings being detected by the virus scanners. There is a chance that the binaries we have predicted to be false positives may also have been detected by yara string rules. Because we aren't able to see exactly what strings the yara rules are looking for, we can't say exactly which of them may be causing false positives. We can, however, recommend that the string rules be examined more closely and perhaps modified to lower the number of possible false positives.

Examining the snort rules makes us look at different aspect of the Citadel trojan. Snort rules are mainly based on network traffic. To get the best idea of what snort rules could affect false positives we would need to analyze pcaps included with the binaries. Unfortunately, none of the binaries that used the rules we listed above contained pcaps. We can see that snort rules look for outbound and inbound traffic of the Citadel control panel. This is a strong rule since it is specific to the citadel trojan. This is similar to the traffic from VNC, and video that the citadel snort rules look for. These services are exploited with the citadel trojan in order to obtain personal credentials like banking information. We assume that these rules are strong and most likely do not cause false positives as the actions of citadel are very specific.

The IOC rules give us a good mixture of rules from both the network and application side of things. It lists specific IP addresses to look for that are known to be related to citadel and zeus command servers. However just blocking a specific IP is not that strong of a rule. An attacker could easily spoof the IP address in which the bot is controlled from or responds to. This would essentially make the rule obsolete. If an IP that was being spoofed and was put into these IOC rules, it could end up that a legitimate IP address was blocked by these rules.

Overall this assignment was a difficult task. Unfortunately, Citadel is still an active trojan that continues to grow. Because of this, there are not many rules that are published to the public. This is because it would be easy for these malware developers to just build around the rules. The Citadel trojan continues to be heavily analyzed by malware researchers. There are few papers and data about it, which makes it difficult to get specifics about rules. Having no verified false positives made it extremely hard to get any kind of baseline to compare with our samples. Because of this we really have no way of definitively proving that any of these are false positives. We can only use the data we were able to obtain to give our best prediction of possible false positives. From this assignment we learned a lot about the malware rule writing process. It is something that takes a lot of time and research, usually by a small niche of

people that do not work for any one company. There is still a lot of work to be done regarding the Citadel trojan, and we hope that some of our work may be useful in helping that cause.