COMP 3010: Task 1 Lab Report

Introduction

Between 2006 and 2010 alone, the number of DDoS attacks nearly doubled on a year-by-year basis [8]. With this threat not slowing down [9], its critical security analysts maintain a high state of alert over traffic transiting through their networks. In this report, I will showcase my recreation of a snort alert as a rule and investigation into an infected system's logs for the culprit malware and method of entry.

Methodology

Part 1 - Understanding a Snort alert and re-create a rule to act similarly

Starting with the task_snort_alerts file, I guessed it would lead to clues relating to what the computer in the pcap file was infected with. I began by recording each unique alert message [10] and evaluated them against how much they would help me in the 2nd part of the assignment after recreation as a rule. Alerts classified as MALWARE-CNC drew interest because they likely infected the computer in the first place. Based off this (and the large quantity in the file), I chose the "MALWARE-CNC Win.Trojan.Pushdo variant outbound connection" alert to recreate. "Win.Trojan.Kazy" was also present but dismissed as it's quantity in the alert file was minimal (and later revealed to be an alias of Pushdo [12]).

Using grep, I filtered the Pushdo alerts out and analysed them for common patterns. I noticed that the flagged logs originated from the home network (192.168.1.*) over port 80. Although I used HOME_NET and EXTERNAL_NET in my answer, they can be replaced with 192.168.1.0/24 and any, respectively for the same result. Researching the trojan, I discovered it's designed to download and execute other malware on Windows hosts. These calls were likely attempts to communicate with the command-and-control server hosting the malware. The research [1-6] was crucial in reverse engineering the snort alert, drawing assumptions and patterns from existing citations. Furthermore, each flagged log had the ACK flag set which I included in my rule, filtering out any decoy or initial connections.

Part 2 – Investigate how the computer became infected / what the computer was infected with

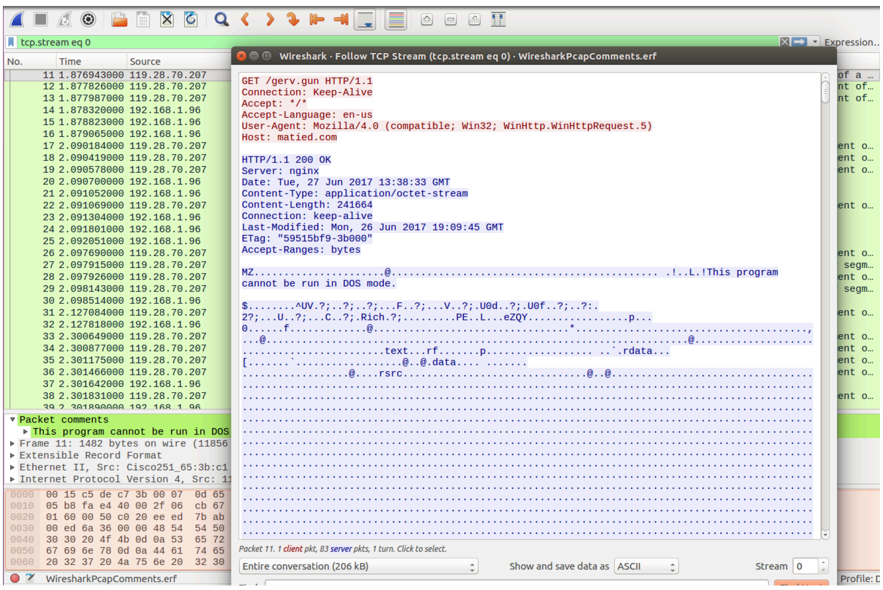
Opening the pcap in Wireshark, I noticed a http request to obtain “gerv.gun” after an initial DNS resolution and SYN/ACK to matied.com. The domain I later discovered to be malicious and is referred to as Matied in the remainder of the report. The request contains the same Accept and Connection headers as Pushdo (and even User-Agent in the case of my references).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000…	192.168.1.96	192.168.1.1	DNS	74	Standard query 0x860f A matied.com
2	0.2010…	192.168.1.1	192.168.1.96	DNS	90	Standard query response 0x860f A matied.com A 119.28.70.207
3	0.2048…	192.168.1.96	119.28.70.207	TCP	70	49184 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
4	0.4169…	119.28.70.207	192.168.1.96	TCP	70	80 → 49184 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1424 SACK_PERM=1 WS=128
5	0.4174…	192.168.1.96	119.28.70.207	TCP	64	49184 → 80 [ACK] Seq=1 Ack=1 Win=66816 Len=0
6	0.4176…			ERF	88	Provenance Metadata Record
7	0.4176…	192.168.1.96	119.28.70.207	HTTP	234	GET /gerv.gun HTTP/1.1
8	0.6326…	119.28.70.207	192.168.1.96	TCP	58	80 → 49184 [ACK] Seq=1 Ack=177 Win=30336 Len=0

Researching gerv.gun, I found a study into Ursnif malware that shows similarities to my findings. Realising that it’s a downloader trojan (of a similar malware family to Pushdo), I set Wireshark’s filter to “http.request” and found more requests for confirmed malicious or suspicious files. t64.bin looked suspicious at first but was found to be benign [11].

No.	Time	Source	Destination	Protocol	Length	Info
7	0.4176…	192.168.1.96	119.28.70.207	HTTP	234	GET /gerv.gun HTTP/1.1
320	317.98…	192.168.1.96	119.28.70.207	HTTP	564	POST /auth/ajax/847598782/?min=data HTTP/1.1 (application/x-www-form-urlencoded)
327	318.83…	192.168.1.96	119.28.70.207	HTTP	666	POST /auth/min/828949448/ HTTP/1.1 (application/x-www-form-urlencoded)
337	320.00…	192.168.1.96	145.131.10.21	HTTP	204	GET /oud/trow.exe HTTP/1.1
693	321.89…	192.168.1.96	143.95.151.192	HTTP	206	GET /wp.exe HTTP/1.1
894	326.48…	192.168.1.96	59.106.164.230	HTTP	173	GET /img/t64.bin HTTP/1.1

Next, I wanted to discover what happened after the initial connection was formed. I utilised Wireshark’s “Follow TCP” feature, allowing me to view unique streams of TCP data stemming from the original gerv request:



- Packet 11, containing HTTP headers from Matied. The Connection and Content-Type headers were set similarly to Pushdo headers. The response contained “This Program cannot be run in DOS mode.”, a phrase used by Pushdo in its reconnaissance efforts [5].
- Packets 70-79, containing initialisation and memory errors alongside (random?) charsets, occluding to garbage data of a successful buffer overflow or format string attack on the underlying website. Just after the errors are a set of Windows API calls returning memory and system info, exactly what a recon malware would be doing.

```
Wireshark · Follow TCP Stream (tcp.stream eq 0) · WiresharkPcapComments.erf
0g..W..=...........................  .........................................Iq..I.`B.
`B..Y....n..Y....n..tan.cos.sin.modf....floor...ceil....atan....exp10...acos....asin....log.log1
0...exp.pow.CorExitProcess..mscoree.dll.runtime error ..
..TLOSS error
...SING error
....DOMAIN error
..R6034
An application has made an attempt to load the C runtime library incorrectly.
Please contact the application's support team for more information.
......R6033
- Attempt to use MSIL code from this assembly during native code initialization
This indicates a bug in your application. It is most likely the result of calling an MSIL-
compiled (/clr) function from a native constructor or from DllMain.
..R6032
- not enough space for locale information
......R6031
- Attempt to initialize the CRT more than once.
This indicates a bug in your application.
..R6030
- CRT not initialized
..R6028
- unable to initialize heap
....R6027
- not enough space for lowio initialization
....R6026
- not enough space for stdio initialization
....R6025
- pure virtual function call
...R6024
- not enough space for _onexit/atexit table
....R6019
- unable to open console device
....R6018
- unexpected heap error
....R6017
- unexpected multithread lock error
....R6016
- not enough space for thread data
.
This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.
```

```
Wireshark · Follow TCP Stream (tcp.stream eq 0) · WiresharkPcapComments.erf
This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.
...R6009
- not enough space for environment
.R6008
- not enough space for arguments
...R6002
- floating point support not loaded
....Microsoft Visual C++ Runtime Library....

......<program name unknown>..Runtime Error!

Program: ... Complete Object Locator'... Class Hierarchy Descriptor'.... Base Class Array'..
Base Class Descriptor at (. Type Descriptor'...`local static thread guard'.`managed vector copy
constructor iterator'..`vector vbase copy constructor iterator'....`vector copy constructor
iterator'..`dynamic atexit destructor for '....`dynamic initializer for '..`eh vector vbase copy
constructor iterator'.`eh vector copy constructor iterator'...`managed vector destructor
iterator'....`managed vector constructor iterator'...`placement delete[] closure'....`placement
delete closure'..`omni callsig'.. delete[]... new[]..`local vftable constructor closure'.`local
vftable'.`RTTI...`EH.`udt returning'.`copy constructor closure'..`eh vector vbase constructor
iterator'..`eh vector destructor iterator'.`eh vector constructor iterator'....`virtual
displacement map'..`vector vbase constructor iterator'.`vector destructor iterator'....`vector
constructor iterator'...`scalar deleting destructor'....`default constructor closure'...`vector
deleting destructor'....`vbase destructor'..`string'....`local static
guard'....`typeof'....`vcall'.`vbtable'...`vftable'...^=..|=..&=..<<=.>>=.%=../=..-=..
+=..*=..||..&&..|...^...~...()..,...>=..>...<=..<...%.../...->*.&...+...-...--..++..*...-
>..operator....[]..!=..==..!...<<..>>..=... delete.
new....__unaligned.__restrict..__ptr64.__clrcall...__fastcall..__thiscall..__stdcall...__pascal.
...__cdecl.__based(......A...A...A...A...A.t.A.h.A.`.A.T.A.H.A...A...A.l.A.X.A.8.A...A.@.A.8.A.
4.A.0.A.,.A.(.A.$.A.
.A...A...A...A...A...A...A...A...A...A...A...A...A...A...A...A...A...A...A...A...A...A...A...A..
.A...A...A...A...A...A...A...A...A...A...A.|.A.t.A.h.A.P.A.D.A.
0.A...A...A...A...A...A.l.A.P.A.,.A...A...A...A...A...A...A...A.x.A.p.A.d.A.T.A.
8.A...A...A...A...A.t.A.X.A.4.A...A...A...A...A...A.........
..
.................. !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~.........?
_nextafter.._logb..._yn._y1._y0.frexp...fmod...._hypot.._cabs...ldexp...fabs....sqrt....atan2...
tanh....cosh....sinh....InitializeCriticalSectionAndSpinCount...kernel32.dll....GetProcessWindow
Station.GetUserObjectInformationA...GetLastActivePopup..GetActiveWindow.MessageBoxA.USER32.DLL..
```

Finally, I decided to run snort (with my custom rule) to reveal what the machine was running at the time the logs were captured.

```
student@snortvm:~/Documents/Coursework$ snort -c /etc/snort/snort.conf -r CNET349SLTask3-pcap.pcap -l LogSpam/
```

The resultant HTML reflects a benign website containing input fields [14] encapsulated in a JSON object, likely where the overflow was deployed.

Results

Part 1

```
# Matches older variants of Pushdo with unencrypted body
alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg: "MALWARE-CNC Win.Trojan.Pushdo variant outbound connection"; classtype:trojan-activity
; flow:to_server,established; flags: A; sid: 29890; rev: 7; content: "GET"; http_method; content: "Connection|3A| Keep-Alive"; http_head
er; nocase; content: "Content-Type|3A| application|2f|octet-stream"; http_header; nocase; content: "Accept|3A| |2a 2f 2a|"; http_header;
 nocase; content: "/s_"; pcre: "/\/s_[0-9]*_[0-9]*\?m=[0-9]?&a=[0-9]&r=[0-9]?&hdd=[0-9a-dA-Z]*&fs=[0-1]+&gen=[0-9]*&os=[0-9]*/P";)

# Matches newer variants of Pushdo with obfuscated HTTP Body
alert tcp $HOME_NET any -> $EXTERNAL_NET 80 (msg: "MALWARE-CNC Win.Trojan.Pushdo variant outbound connection"; classtype:trojan-activity
; flow:to_server,established; flags: A; sid: 29891; rev: 7; content: "GET"; http_method; content: "Connection|3A| Keep-Alive"; http_head
er; nocase; content: "Content-Type|3A| application|2f|octet-stream"; http_header; nocase; content: "Accept|3A| |2a 2f 2a|"; http_header;
 nocase; content: "/40"; pcre: "/\/40[eE]8[0-9A-Z0-9]+6C[0-9]+66[0-9]+76[0-9]+EB[0-9]+30[0-9A-Z]+/P";)
```

Later variants of Pushdo contain an obfuscated payload so I developed two snort rules with slightly differing sid's (messages and class types are unchanged). Breaking down what the rules do:

- "flow" field is set to pick up outbound requests that have reached their destinations.
- "http_methods" filter that only flags GET requests (Pushdo requests data, it doesn't respond to it).
- Several "http_headers" Pushdo will likely have set in its requests. Some use a mix of Hex and ASCII data (if the char needs to be escaped):
    - "Connection: Keep-Alive". Maintains a connection so subsequent malware downloads complete quicker and stealthier.
    - "Content-Type: application/octet-stream". Present in both variants but especially important for later ones, they stream data to Matied inside binarized, formatted objects.

  - “Accept: */*”. Malware that is Matied delivers can be in any media format.
- Regex that searches “http_client_body” (using the P flag) for a set of parameters unique to Pushdo requests. One rule covers the older plaintext format, the other the newer:

```
/ \/s_[0-9]*_[0-9]*\?m=[0-9]?&a=[0-9]&r=[0-9]?&hdd=[0-9a-dA-Z]*&fs=[0-1]+&gen=[0-9]*&os=[0-9]* / gm

TEST STRING

GET /s_60_16909060?
m=3&a=1&r=1&hdd=2020202057202d4443574d413141323735383234&fs=1&
gen=1&os=87489295757392957 HTTP/1.0
```

```
/ \/40[eE]8[0-9A-Z0-9]+6C[0-9]+66[0-9]+76[0-9]+EB[0-9]+30[0-9A-Z]+ / gm

TEST STRING

GET
/40E800080289ED75F373CD0C6C0000001266000000007600000173EB00053
04561717F HTTP/1.0
```

Part 2

Based off of all of the evidence I collected, I conclude that the computer was infected with a downloader trojan (likely part of the Pushdo family, probably Ursnif), which performed reconnaissance operations via the gerv binary. After retrieving basic system details using the Windows API, it proceeded to download tailored malware from Matied (trow.exe) based off the feedback received from gerv.

As for how the computer was initially infected, the logs do not stretch beyond the initial DNS resolution so it’s likely the computer was infected by a drive-by download on matied.com or it was already infected. It’s possible the frontend application contains a XSS vulnerability, allowing for code execution and buffer overflow of an input field (there’s no user input sanitisation on any of the fields). This theory is supported by the alert file which reports multiple buffer overflow and unescaped, obfuscated Javascript attempts inside JumpToIt() function.

Discussion

For Part 1, my solution could be better optimised against a dynamic malware instead of assuming Pushdo’s behaviour is static. An argument could be made for setting snort to scan all http ports, not just 80. The authors of Pushdo would likely adapt and use an alternative route if consistently blocked over port 80. From my research, the user agent is Mozilla across all systems, so an argument could also be made to include that too.

Furthermore, more attention should have been paid to all alerts in the file, as some offered clues on where to examine in the pcap. For example, knowing Pushdo and Kazy are aliases of the same malware [12] early on would have been useful in tracking the infection’s progress.

As for Part 2, I didn’t make use of tcpdump to evaluate hex dumps of packets from Wireshark. Doing so on one of the packets marked may have revealed the source of infection without speculations. Although my use of Wireshark filters to track related data was useful, sometimes it masked the response to said data, leading to longer investigation times.

Conclusion

In conclusion, the damage malware can do that hasn't been intercepted is great, as is the effort to triage and identify the malware in question using logs alone. Modern websites and machines are still vulnerable to classic security flaws if user input is not sanitised and IDS's are not adaptable to new situations.

References

1) Wikipedia (2020). *List of HTTP header fields* [Online]. Subsection *Request Fields.* Available at https://en.wikipedia.org/wiki/List_of_HTTP_header_fields (Accessed 21/11/20).
2) Brandon Stultz (2016). *Introduction to Snort Rule Writing* [Online]. Available at https://www.ciscolive.com/c/dam/r/ciscolive/emea/docs/2016/pdf/DevNet-1693.pdf (Accessed 21/11/20).
3) Malpedia (2020). *Pushdo* [Online]. Available at https://malpedia.caad.fkie.fraunhofer.de/details/win.pushdo (Accessed 21/11/20).
4) Joe Stewart (2007). *Pushdo - Analysis of a Modern Malware Distribution System* [Online]. Available at https://www.secureworks.com/research/pushdo (Accessed 21/11/20).
5) Alice Decker, David Sancho, Loucif Kharouni, Max Goncharov, Robert McArdle (2009). *Pushdo / Cutwail Botnet - A study of the Pushdo / Cutwail Botnet* [Online]. Available at https://www.trendmicro.de/cloud-content/us/pdfs/business/white-papers/wp_study-of-pushdo-cutwail-botnet.pdf (Accessed 21/11/20).
6) Hybrid Analysis (2017). *gerv.gun* [Online]. Available at https://www.hybrid-analysis.com/sample/0931537889c35226d00ed26962ecacb140521394279eb2ade7e9d2afcf1a7272?environmentId=100 (Accessed 21/11/20).
7) Kapli Kulkarni (2018). *Threat Hunting – Malspam –Japan Office Infected* [Online]. Section *Scenario.* Available at https://resources.infosecinstitute.com/topic/threat-hunting-malspam-japan-office-infected (Accessed 21/11/20).
8) Waleed Bul'ajoul, Anne James, Mandeep Pannu (2015). *Improving network intrusion detection system performance through quality of service configuration and parallel technology* [Online]. *Volume 81, Issue 6, Abstract – Introduction.* Available at https://www.sciencedirect.com/science/article/pii/S0022000014001767 (Accessed 21/11/20).

9) Paul Nicholson (2020) *AWS hit by Largest Reported DDoS Attack of 2.3 Tbps* [Online]. Available at https://www.a10networks.com/blog/aws-hit-by-largest-reported-ddos-attack-of-2-3-tbps (Accessed 21/11/20).
10) Morgan Davies (2020). *Task1_Notes.txt* [Online]. Available at https://liveplymouthac-my.sharepoint.com/:t:/g/personal/morgan_davies_students_plymouth_ac_uk/EULFu9vV3EpChejMKzjdvZMBxEaQJwp03c_dNt2y_8DhKg?e=pw6npN (Accessed 22/11/20).
11) Hybrid analysis (2018). *t64.bin* [Online]. Available at https://www.hybrid-analysis.com/sample/a125b2e41ce61f23bf16c5bc143809e4dac3f5822ee8b48b22dfe6fe16e35ef6 (Accessed 23/11/20).
12) Sophos (2013). *Troj/Pushdo-BB* [Online]. Available at https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/Troj~Pushdo-BB.aspx (Accessed 23/11/20).
13) Hybrid analysis (2019). *trow.exe* [Online]. Available at https://www.hybrid-analysis.com/sample/94a0a09ee6a21526ac34d41eabf4ba603e9a30c26e6a1dc072ff45749dfb1fe1?environmentId=120 (Accessed 23/11/20).
14) Pohl Food Service (2020). *Pohl Food Service* [Online]. Available at http://www.pohlfood.com/index.html (Accessed 24/11/20).