AFS Volume Corruption Due To Defect In Juniper Switch OS

Yadavendra Yadav
Todd DeSantis
John P Janosik
Indira V Khopde

Agenda

- Root Cause of Data Corruption
- Trigger for Data Corruption
- Problem Reported By Client
- Initial Troubleshooting
- Identifying Corruption Patterns
- Recreating The Issue
- Troubleshooting AFS RX Layer
- Debugging Network Traffic
- Juniper Defect Analysis
- Examples of Similar Data Corruption

Root Cause of Data Corruption

After troubleshooting, we found that the data corruption was caused due to one of the defects in Juniper OS.

http://www.juniper.net/techpubs/en_US/junos12.3/information-products/topic-collections/release-notes/12.3/topic-81613.html

"The non-first IP fragments containing UDP payload may be mistakenly interpreted as PTP packets if the following conditions are met: - the byte at the offset 9 in the IP packet contains 0x11 (decimal 17) - UDP payload - the two bytes at the offset 22 in the IP packet contain the value 0x01 0x3f (decimal 319; byte 22=0x01 and byte 23=0x3f) - PTP protocol The mis-identification of the packet as PTP will trigger the corruption of the fragment payload. PR1006718: This issue has been resolved."

Root Cause of Data Corruption (cont...)

Juniper OS defect gets triggered under following conditions:

- Packet should be non-first IP fragment.
- Byte at 9th offset in IP packet should contains 0x11.Which is transport protocol type and 0x11 corresponds to UDP.
- In IP packet at offset 22 we should have 0x01 0x3f.Which is a destination port in UDP header and 0x01 0x3f i.e. port 319 corresponds to a destination port for PTP event message (http://wiki.hevs.ch/uit/index.php5/Standards/Ethernet_PTP/frames).

Trigger for Data Corruption

- There was a specific .gz format file in a volume, which had the data corruption. We never saw corruption in an ascii file.
- Volume was moved between two separate physical server. In case source and destination fileserver were on same physical system, data corruption didn't happen
- If the volserver was started with -nojumbo option, the data corruption didn't happen
- Jumbo packets should get fragmented while moving from source to destination server. This means that Path MTU from source to destination should be less than the size of jumbo packets

Problem Reported By Client

- User reports possible corruption because unzip of a .gz file was failing.
- Several more users started to report similar problems.
- Corrupted data is isolated to volumes in our new Lab.
- Stopped Data Migration to the new servers on the new Lab.

Problem Reported By Client (cont...)

- Started Investigation on how the data was migrated to the file servers in the new Lab. User reported possible corruption because unzip of a .gz file was failing.
- Suspected “Vice Partition Virtualization” Feature that transferred all vice partitions to the new fileserver via SAN Data Sharing.
- Only found corrupted files on volumes that were “vos moved” by the Balancing Tool.
- Data Corruption never happened when "vos move" operation was tried between two fileserver on the SAME physical machine.
- Data Corruption happened only when "vos move" operation was tried between two fileserver on DIFFERENT physical machines.
- So from above test it was clear that “vos” transaction needs to touch the physical network in the new Lab, for data corruption to happen.

Problem Reported By Client (cont...)

	Old Server – Old Lab	New Server– Old Lab	New Server – New Lab
Old–Old	OK	OK	OK
New–Old	OK	OK	BAD
New –New	OK	BAD	BAD

- Data Corruption was found on New Servers.
- Three machine configurations:
  - Old Server in Old Lab
  - New Server in Old Lab
  - New Server In New Lab

Initial Troubleshooting

- Ruled out VPV Feature as the cause of the corruption.
- Found that "vos move" caused the corruption and Load Balancing Tool was the only thing moving volumes inside the new lab.
- Shutdown Load Balancing Tool for the new lab.
- Found that new server configuration was NOT using –nojumbo.
- Stopped and Started new servers to use –nojumbo.
-
- Verified that “vos move” was no longer corrupting volumes.
- Sites were now stable.

Initial Troubleshooting (cont...)

- Engaged Network Team and Network Admins.
- Anything different with network for the new lab v/s old lab or other labs at other sites.
- Network team did not indicate any issues and they continued to test.
- We started out to determine if AFS was causing the corruption or if it was something related to the network.
-
- Really, we were trying to prove that AFS was NOT causing the problem.

Identifying Corruption Patterns

- Determine what is corrupted in these large gizpped tar images.
- For this we used "od" tool and dumped the content of a file in hex format "od –h”
- We used "diff" tool to compare the data between good and bad version of a file.

Below are sample steps:

Step 1 : od -h <good version of .gz file> > dump-good-version

Step 2 : od -h <bad version of .gz file> > dump-bad-version

Step 3 : diff -h dump-good-version dump-bad-version \
> diff-between-good-bad-version

Identifying Corruption Patterns (cont...)

Found patterns of Corruption. Let's take one of the Corruption pattern:

Good Data :

> 2655454560 54fe 3e00 588b 1200 ec01 3f00 2875 1200
> 2655454600 14f8 3e00 588b 1200 acfb 3e00 2875 1200

Bad Data :

< 2655454560 54fe 3e00 588b 1200 ec01 3f00 285c 6300
< 2655454600 14f8 3e00 588b 1200 acfb 3e18 d775 1200

Pattern 1

- 0x7512 was changed to 0x5c63
- 013f <2bytes> <2 bytes corruption>

Pattern 2

- 0x7512 was changed to 0x5c63. Difference 0x7512 - 0x5c63 = 0x18AF
- 0x0028 was changed to 0x18d7. Difference 0x0028 - 0x18d7 = - 0x18AF

Recreating the Issue

- Tried to use ASCII files to reproduce the Data corruption - Data corruption never happened
- Tried to create a volume with just copies of the good data and moved them - Sometimes we saw corruption
- Sometimes the file was corrupted in 2 places, 4 places, 7 places, never in the same location
- Enabling jumbograms was one the conditions to simulate the issue
- Adding/deleting files in volumes had an effect on Data Corruption

Recreating the Issue (cont...)

- Finally found a reproducible case on a subset of the corrupted data that we could reproduce 100% of the time via "vos restore" and 99.99% via "vos move" operation.
- Corruption happened at same place and for same number of bytes.
- Discussed with Jeffrey Altman.
- Tested MTU negotiation . Did not see issue with MTU negotiation.
- Added secure connections. With this, data corruption never happened. A secure connection turns off AFS jumbograms and also encrypts the data.

Troubleshooting AFS RX Layer

- Added a special RX logging inside a RX layer, to investigate content of Sent & Received Packets

- Added rx debugging options for Volserver and Vos. Similar to "fileserver -rxdbg".

Conclusions :

Once RX data was collected we tried to grep for Good and Bad patterns in RX debug logs.

- ✔ On Sender RX debug logs we never saw corrupted pattern.
- ✔ On Receiver RX debug logs we saw corrupted pattern.

Debugging Network Traffic

- Took tcpdump of network traffic on the Sender and Receiver side
- Tried to find the corrupt patterns in the packet using Wireshark.
- Analyzed Packet content to verify:
  - Whether on Sender side Corrupted Data was sent.
  - Whether on Receiver side Corrupted Data was received.

Debugging Network Traffic (cont...)

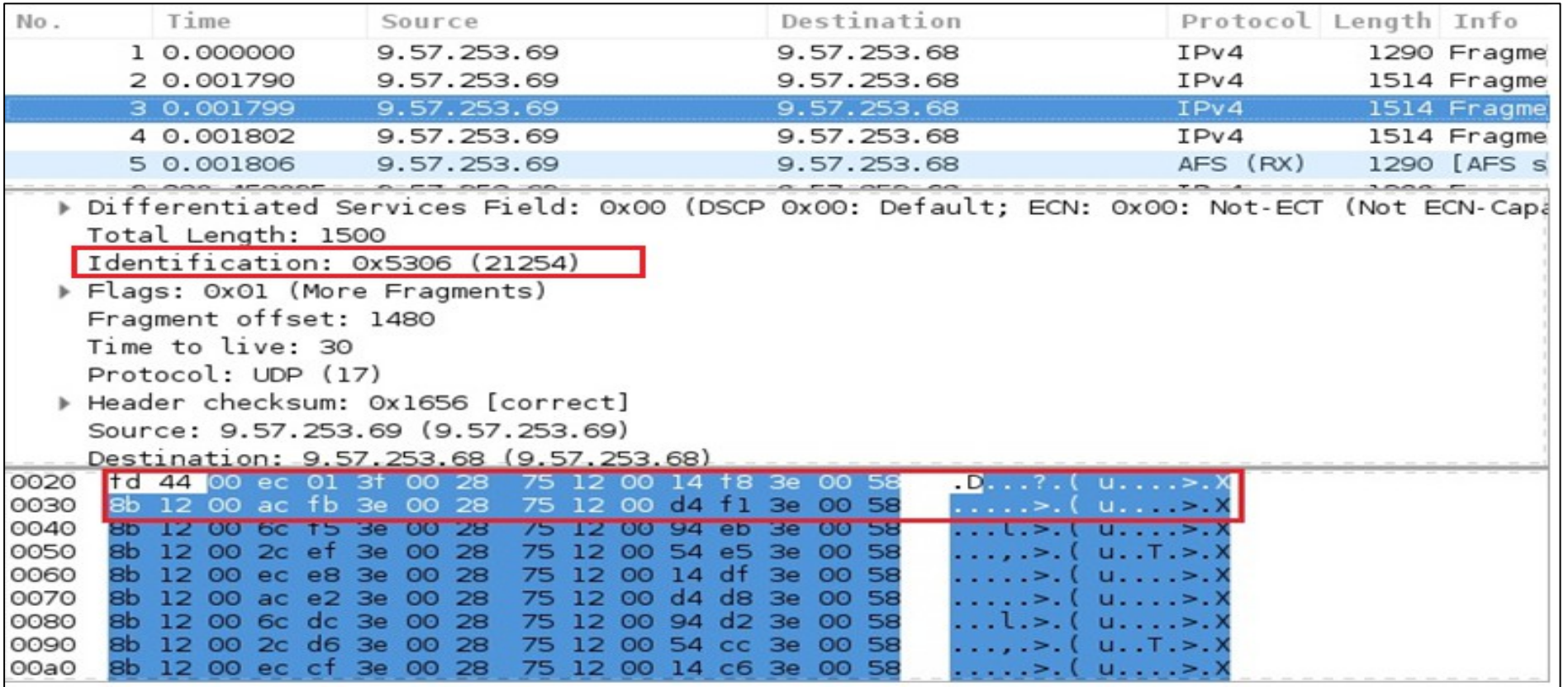
Sender TCPDUMP analysis :

Good Data

- 2655454560 54fe 3e00 588b 1200 ec01 3f00 2875 1200
- 2655454600 14f8 3e00 588b 1200 acfb 3e00 2875 1200

Bad Data

- 2655454560 54fe 3e00 588b 1200 ec01 3f00 285c 6300
- 2655454600 14f8 3e00 588b 1200 acfb 3e18 d775 120



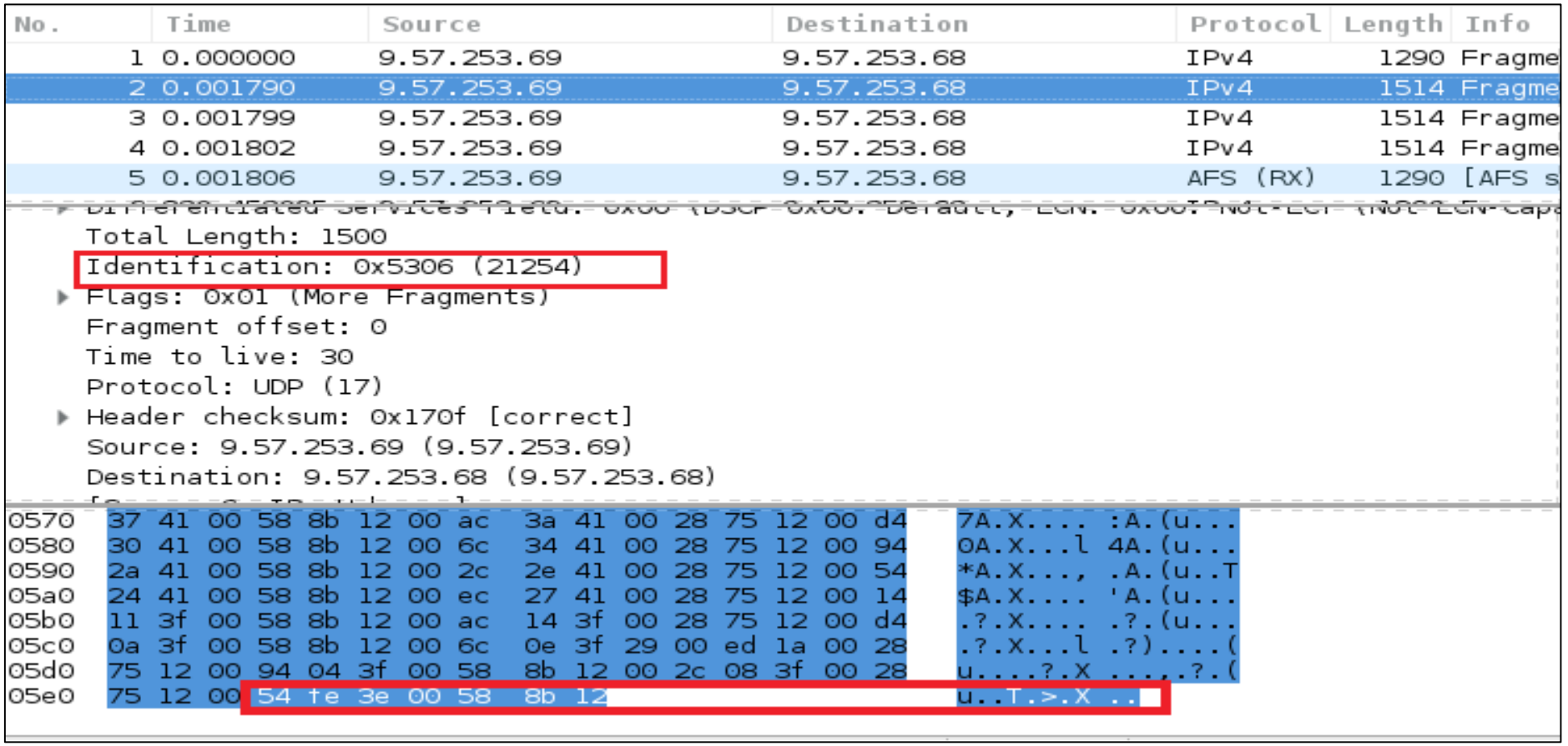
Debugging Network Traffic (cont...)

Good Data — Sender TCPDUMP analysis :

- 2655454560 54fe 3e00 588b 1200 ec01 3f00 2875 1200
- 2655454600 14f8 3e00 588b 1200 acfb 3e00 2875 1200

Bad Data

- 2655454560 54fe 3e00 588b 1200 ec01 3f00 285c 6300
- 2655454600 14f8 3e00 588b 1200 acfb 3e18 d775 120



Debugging Network Traffic (cont...)

Good Data

Receiver TCPDUMP Analysis :

- 2655454560 54fe 3e00 588b 1200 ec01 3f00 2875 1200
- 2655454600 14f8 3e00 588b 1200 acfb 3e00 2875 1200

Bad Data

- 2655454560 54fe 3e00 588b 1200 ec01 3f00 285c 6300
- 2655454600 14f8 3e00 588b 1200 acfb 3e18 d775 1200

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	9.57.253.69	9.57.253.68	IPv4	1290	Fragme
2	0.001773	9.57.253.69	9.57.253.68	IPv4	1514	Fragme
3	0.001781	9.57.253.69	9.57.253.68	IPv4	1514	Fragme
4	0.001787	9.57.253.69	9.57.253.68	IPv4	1514	Fragme
5	0.001792	9.57.253.69	9.57.253.68	AFS (RX)	1290	[AFS s

```
Total Length: 1500
Identification: 0x5306 (21254)
▸ Flags: 0x01 (More Fragments)
Fragment offset: 1480
Time to live: 30
Protocol: UDP (17)
▸ Header checksum: 0x1656 [correct]
Source: 9.57.253.69 (9.57.253.69)
Destination: 9.57.253.68 (9.57.253.68)
[Source GeoIP: Unknown]
```

```
0020  fd 44 00 ec 01 3f 00 28  5c 63 00 14 f8 3e 00 58   .D...?.( \c...>.X
0030  8b 12 00 ac fb 3e 18 d7  75 12 00 d4 f1 3e 00 58   .....>.. u....>.X
0040  8b 12 00 6c f5 3e 00 28  75 12 00 94 eb 3e 00 58   ...l.>.( u....>.X
0050  8b 12 00 2c ef 3e 00 28  75 12 00 54 e5 3e 00 58   ...,.>.( u..T.>.X
0060  8b 12 00 ec e8 3e 00 28  75 12 00 14 df 3e 00 58   .....>.( u....>.X
0070  8b 12 00 ac e2 3e 00 28  75 12 00 d4 d8 3e 00 58   .....>.( u....>.X
```

Debugging Network Traffic (cont...)

- One weekend, Yadav and I are discussing the corruption and running tests and we identify that every time we see the corrupted data, that we also see the 01 3f bytes near our corrupted data.
- We verify that in all of our test cases, we see this pattern.
- We start to run more tests and all of a sudden, we can no longer reproduce the problem.
- We tried for hours to reproduce the problem and no luck.
- Then I received an email from the site Network Admin .....

Juniper Defect Analysis

- Network Admin decide to upgrade the Juniper switches in the new Lab. Once the upgrade was done Data Corruption never happened.
- Recommended the Network Team to scan the Release Notes for the Juniper upgrade. They finally found a defect that might have resulted in the data corruption.
- JunOS defect gets triggered under following three conditions:
  - Packet should be non-first IP fragment.
  - Byte at 9th offset in IP packet contains 0x11 (Which is actually transport protocol type and 0x11 corresponds to UDP).
  - In IP packet at offset 22 we should have 0x01 0x3f.

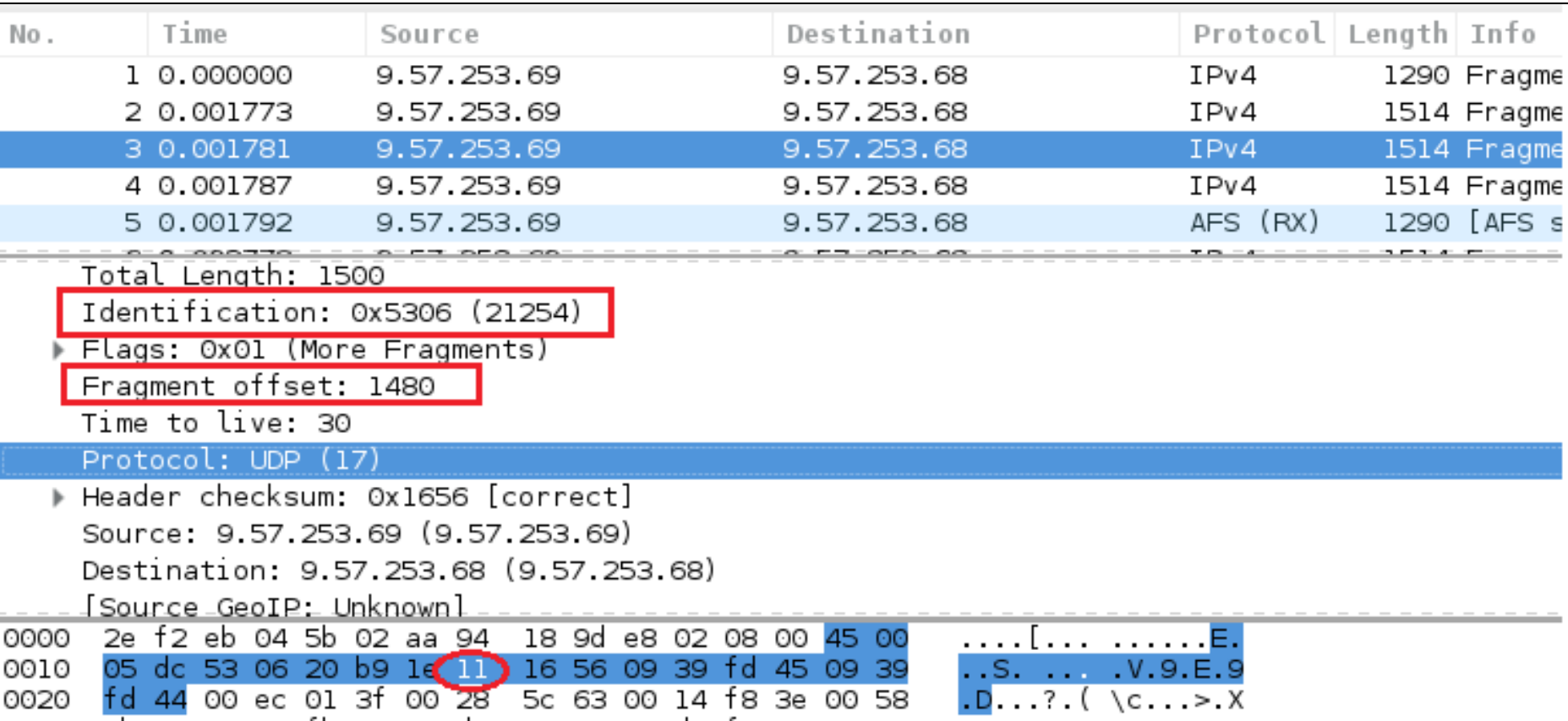
Juniper Defect Analysis (cont...)

Validate the Packet to verify if Juniper OS defect has caused Data Corruption.

Condition 1 & 2:

(1) Packet should be non-first IP fragment.

(2) Byte at 9th offset in IP packet contains 0x11 (Which is actually transport protocol type and 0x11 corresponds to UDP).



Juniper Defect Analysis (cont...)

Condition 3:

In IP packet at offset 22 we should have 0x01 0x3f.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	9.57.253.69	9.57.253.68	IPv4	1290	Fragme
2	0.001773	9.57.253.69	9.57.253.68	IPv4	1514	Fragme
3	0.001781	9.57.253.69	9.57.253.68	IPv4	1514	Fragme
4	0.001787	9.57.253.69	9.57.253.68	IPv4	1514	Fragme
5	0.001792	9.57.253.69	9.57.253.68	AFS (RX)	1290	[AFS s

```
▸ Header checksum: 0x1656 [correct]
  Source: 9.57.253.69 (9.57.253.69)
  Destination: 9.57.253.68 (9.57.253.68)
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
  Reassembled IPv4 in frame: 5
▾ Data (1480 bytes)
    Data: 00ec013f00285c630014f83e00588b1200acfb3e18d77512...
    [Payload MD5 hash: 383530e239d54d5e3cfa69e05fa18b81]
    [Length: 1480]
```

```
0000  2e f2 eb 04 5b 02 aa 94  18 9d e8 02 08 00 45 00   ....[... ......E.
0010  05 dc 53 06 20 b9 1e 11  16 56 09 39 fd 45 09 39   ..S. ... .V.9.E.9
0020  fd 44 00 ec 01 3f 00 28  5c 63 00 14 f8 3e 00 58   .D...?.( \c...>.X
0030  8b 12 00 ac fb 3e 18 d7  75 12 00 d4 f1 3e 00 58   .....>.. u....>.X
```

Example: Directory Corruption

- Several client machines get bad “ls -l”
- Points to a corrupted filename.
- All clients isolated to a specific lab subnet.

```
ls: 0653-341 The file ./builtin_M-^LWrorif.gold.graph does not exist.
total 0
1255747638 -rw-r--r--   1 1544     staff         407 Jul 10 2013  builtin_aselect.gold
1255746890 -rw-r--r--   1 1544     staff           0 Jul 10 2013  builtin_aselect.gold.graph
1255747470 -rw-r--r--   1 1544     staff         665 Jul 10 2013  builtin_debug.gold
1255747774 -rw-r--r--   1 1544     staff           0 Jul 10 2013  builtin_debug.gold.graph
1255747328 -rw-r--r--   1 1544     staff         260 Jul 10 2013  builtin_debugif.gold
```

Example: Directory Corruption

- Strange characters in filename, point to data corruption
- Diffs of good/bad “ls -li” output

```
[] diff lsli.bad lsli.good
1d0
< ls: 0653-341 The file ./builtin_Wrorif.gold.graph does not exist.
13a13
> 1255747694 -rw-r--r--    1 1544     staff          0 Jul 10 2013  builtin_errorif.gold.graph
```

Example: Directory Corruption (cont...)

- Found cache Vfiles of good/bad directory.
- Converted to AFS Dir output.
- Took Diff of files.
- Shows name and vnode are wrong.

```
[]  diff V41000.lst V43584.lst
496c496
< builtin_errorif.gold.graph    12398   54470
---
> builtin_Wrorif.gold.graph    2441    54470
```

Example: Directory Corruption (cont...)

- Diff of “od -h” versions of the dir files.
- 013f Trigger and 2 sets of corruption.
- Points to the Juniper Bug.

```
[] diff V41000.odh V43584.odh
1682,1683c1682,1683
< 0064540  0100 013f 0000 306e 0000 d4c6 6275 696c
< 0064560  7469 6e5f 6572 726f 7269 662e 676f 6c64
---
> 0064540  0100 013f 0000 0989 0000 d4c6 6275 696c
> 0064560  7469 6e5f 8c57 726f 7269 662e 676f 6c64
```

Questions ??

Thank You