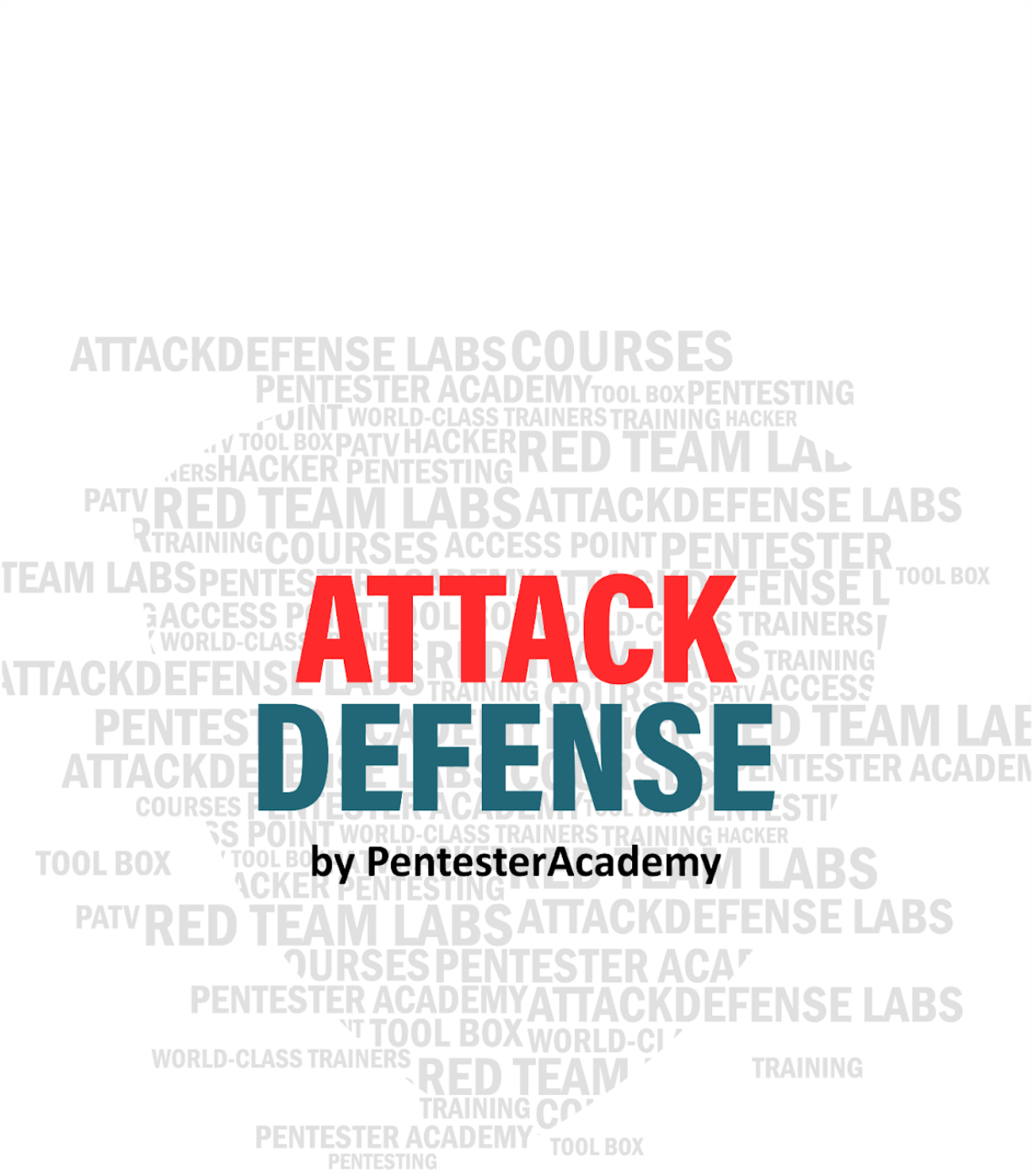
ATTACK
DEFENSE
by PentesterAcademy

Name	Memcache Log Analysis I
URL	https://www.attackdefense.com/challengedetails?cid=1205
Type	Infrastructure Attacks : Memcached

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Stripping the log file:

The message “Kicking LRU crawler off for LRU” is printed for each slab class when the LRU crawler is called. “Nothing left to crawl” messages are printed before LRU crawler sleeps.
The following messages are printed when a memcached command is executed:

1. going from conn_parse_cmd to conn_write
2. going from conn_write to conn_new_cmd
3. going from conn_new_cmd to conn_waiting
4. going from conn_waiting to conn_read
5. going from conn_read to conn_parse_cmd

The above mentioned messages are not required for this lab exercise. Create a new log file by removing the non relevant messages.

Command: cat memcache.log | grep -vE 'Kicking|Nothing|*going*' > memcache-stripped.log

```
root@attackdefense:~#
root@attackdefense:~#
root@attackdefense:~#
root@attackdefense:~# cat memcache.log | grep -vE 'Kicking|Nothing|*going*' > memcache-stripped.log
root@attackdefense:~#
root@attackdefense:~#
```

Q1. Find the total number of connections made to the Memcached server excluding the connection made by the web application.

Answer: 12

Solution:

Since it is specified in the challenge description that the web application maintains an active connection. The socket descriptor will remain the same for the web application connection.

Step 1: Identify the socket descriptor number of the web application.

Command: head -200 memcache-stripped.log

```
root@attackdefense:~# head -200  memcache-stripped.log
slab class   1: chunk size        96 perslab   10922
slab class   2: chunk size       120 perslab    8738
slab class   3: chunk size       152 perslab    6898
slab class   4: chunk size       192 perslab    5461
slab class   5: chunk size       240 perslab    4369
slab class   6: chunk size       304 perslab    3449
slab class   7: chunk size       384 perslab    2730
slab class   8: chunk size       480 perslab    2184
slab class   9: chunk size       600 perslab    1747
slab class  10: chunk size       752 perslab    1394
slab class  11: chunk size       944 perslab    1110
slab class  12: chunk size      1184 perslab     885
slab class  13: chunk size      1480 perslab     708
slab class  14: chunk size      1856 perslab     564
slab class  15: chunk size      2320 perslab     451
slab class  16: chunk size      2904 perslab     361
slab class  17: chunk size      3632 perslab     288
slab class  18: chunk size      4544 perslab     230
slab class  19: chunk size      5680 perslab     184
slab class  20: chunk size      7104 perslab     147
slab class  21: chunk size      8880 perslab     118
slab class  22: chunk size     11104 perslab      94
slab class  23: chunk size     13880 perslab      75
slab class  24: chunk size     17352 perslab      60
slab class  25: chunk size     21696 perslab      48
slab class  26: chunk size     27120 perslab      38
slab class  27: chunk size     33904 perslab      30
slab class  28: chunk size     42384 perslab      24
```

```
slab class  29: chunk size     52984 perslab      19
slab class  30: chunk size     66232 perslab      15
slab class  31: chunk size     82792 perslab      12
slab class  32: chunk size    103496 perslab      10
slab class  33: chunk size    129376 perslab       8
slab class  34: chunk size    161720 perslab       6
slab class  35: chunk size    202152 perslab       5
slab class  36: chunk size    252696 perslab       4
slab class  37: chunk size    315872 perslab       3
slab class  38: chunk size    394840 perslab       2
slab class  39: chunk size    524288 perslab       2
Starting LRU crawler background thread
Starting LRU maintainer background thread
<26 server listening (auto-negotiate)
<27 server listening (auto-negotiate)
LRU crawler thread sleeping
```

```
<28 new auto-negotiating client connection
28: Client using the ascii protocol
<28 set first_name 0 0 5
> NOT FOUND first_name
>28 STORED
<28 set last_name 0 0 5
> NOT FOUND last_name
>28 STORED
<28 set nick_name 0 0 3
> NOT FOUND nick_name
>28 STORED
<28 set address 0 0 14
> NOT FOUND address
>28 STORED
<28 set city 0 0 10
> NOT FOUND city
>28 STORED
<28 set state 0 0 8
> NOT FOUND state
>28 STORED
<28 set zip 0 0 5
> NOT FOUND zip
>28 STORED
<28 set country 0 0 13
> NOT FOUND country
>28 STORED
<28 set password 0 0 7
> NOT FOUND password
>28 STORED
```

```
<28 set email 0 0 24
> NOT FOUND email
>28 STORED
```

```
<28 set title 0 0 30
> NOT FOUND title
>28 STORED
<28 set page_content 0 0 26
> NOT FOUND page_content
>28 STORED
<28 set is_offline 0 0 5
> NOT FOUND is_offline
>28 STORED
<28 set allow_registrations 0 0 5
> NOT FOUND allow_registrations
>28 STORED
<28 set token 0 0 32
> NOT FOUND token
>28 STORED
```

The initial connection had a socket descriptor 28. Since the values were stored through this connection. The connection could be of the web application.

Step 2: As, it is specified in the challenge description that the web application retrieves the key value pairs periodically. Check which connection, retrieves certain key value pairs again and again.

Command: cat memcache-stripped.log | grep get

```
root@attackdefense:~# cat memcache-stripped.log | grep get
<28 get token
<28 get allow_registrations
<28 get is_offline
<28 get title
<28 get page_content
<28 get token
<28 get allow_registrations
<28 get is_offline
<28 get title
<28 get page_content
<28 get token
<28 get allow_registrations
<28 get is_offline
<28 get title
<28 get page_content
<28 get token
<28 get allow_registrations
<28 get is_offline
<28 get title
<28 get page_content
<28 get token
```

```
<28 get allow_registrations
<28 get is_offline
<28 get title
<28 get page_content
<28 get username
<28 get password
<28 get role
```

```
<28 get sessid_11131211224
<28 get token
<28 get allow_registrations
<28 get is_offline
<28 get title
<28 get page_content
<28 get token
```

The value stored in keys token, allow_registrations, is_offline, title and page_content are retrieved again and again by the client connected over 28 socket descriptor. Therefore it can be concluded that socket descriptor 28 corresponds to the client used by the web application.

Whenever a new client connects, the message “new auto-negotiating client connection” is logged in the log file.

Step 3: To identify the total number of connection, search the log for the message “the message “new auto-negotiating client connection”

Command: cat memcache-stripped.log | grep "new auto-negotiating client connection"

```
root@attackdefense:~# cat memcache-stripped.log | grep "new auto-negotiating client connection"
<28 new auto-negotiating client connection
<29 new auto-negotiating client connection
<29 new auto-negotiating client connection
<30 new auto-negotiating client connection
<31 new auto-negotiating client connection
<32 new auto-negotiating client connection
<33 new auto-negotiating client connection
<33 new auto-negotiating client connection
<33 new auto-negotiating client connection
<32 new auto-negotiating client connection
<33 new auto-negotiating client connection
<33 new auto-negotiating client connection
<34 new auto-negotiating client connection
root@attackdefense:~#
```

Excluding the connection made by the web application there were a total of 12 connections.

Since all of the 12 connections were not made by the web application. They can be suspected of performing malicious activity.

Q2. How many slabs did the attacker enumerate using the command “stats cachedump”?

Answer: 3

Solution:

Search for string “stats cachedump” where the socket descriptor is not 28.

Command: cat memcache-stripped.log | grep "stats cachedump" | grep -v "<28"

```
root@attackdefense:~# cat memcache-stripped.log | grep "stats cachedump" | grep -v "<28"
<29 stats cachedump 1 0
<29 stats cachedump 2 0
<29 stats cachedump 3 0
root@attackdefense:~#
```

The key names from 3 slabs were retrieved.

Q3. How many unique key-value pairs were dumped by malicious connections? Assuming that only valid user is the web application.

Answer: 9

Solution:

The unique key value pairs dumped by the connections can be determined with the following command.

Command: cat memcache-stripped.log | grep -v "<28" | grep -A1 "get" | grep 'FOUND KEY' | uniq

grep -v "<28" is used to filter out the commands executed by the web application.
grep -A1 "get" is used to filter the get command and the corresponding output displayed on the next line.

If the key value pair was present, the output will contain the string ‘FOUND KEY <key-name>’
If the key value pair was absent, the output will contain the string “NOT FOUND <key-name>”.

```
root@attackdefense:~# cat memcache-stripped.log | grep -v "<28" | grep -A1 "get" | grep 'FOUND KEY' | uniq
> FOUND KEY password
> FOUND KEY username
> FOUND KEY email
> FOUND KEY token
> FOUND KEY role
> FOUND KEY allow_registrations
> FOUND KEY is_offline
> FOUND KEY sessid_11131211224
> FOUND KEY title
root@attackdefense:~#
```

Total 9 key value pairs were retrieved by the malicious connection

Q4. The web application logic updates a set of essential key-value pairs if any key-value pair of this set is invalidated/deleted/expired. The attacker can leverage such behavior to increase the performance of the web application sometimes leading to DoS. In the provided logs, an attacker has tried to do the same by invalidating a key-value pair. What is the name of that key?

Answer: token

Solution:

Step 1: Identify the keys which were deleted

Command: cat memcache-stripped.log | grep delete

```
root@attackdefense:~# cat memcache-stripped.log | grep delete
<29 delete token
<29 delete email
<29 delete sessid_11131211224
<29 delete token
<30 delete token
<30 delete token
<30 delete token
<30 delete token
<30 delete token
```

```
<30 delete token
<30 delete token
<30 delete token
<30 delete token
<30 delete token
<30 delete token
<30 delete token
root@attackdefense:~#
```

Key token, email, sessid_11131211224 were deleted from the memcached server. Since key token has been deleted many times, it could be the key which upon deletion leads to reinitialization of key value pairs in cache.

After deletion of a key, If the log shows following acticity, it can be concluded that the specified key was responsible for reinitialization of key value pairs.

1. Certain number of key value pairs were set on the cache after deletion of the key
2. The pattern of key deletion followed by setting of key value pair repeats

Step 2: Identify the line number of each delete command.

Command: cat memcache-stripped.log | grep -n delete

```
root@attackdefense:~# cat memcache-stripped.log | grep -n delete
444:<29 delete token
501:<29 delete email
526:<29 delete sessid_11131211224
541:<29 delete token
562:<30 delete token
565:<30 delete token
586:<30 delete token
589:<30 delete token
610:<30 delete token
613:<30 delete token
650:<30 delete token
654:<30 delete token
675:<30 delete token
678:<30 delete token
699:<30 delete token
702:<30 delete token
root@attackdefense:~#
```

Step 3: Check the set of commands executed after each key (the ones mentioned above) was deleted.

View first 50 lines after the key “token” was deleted for the first time.

Command: sed -n '444,494p' memcache-stripped.log

```
root@attackdefense:~# sed -n '444,494p' memcache-stripped.log
<29 delete token
> FOUND KEY token
>29 DELETED
<28 get token
> NOT FOUND token
>28 END
<28 set token 0 0 32
> NOT FOUND token
>28 STORED
<28 set title 0 0 30
> FOUND KEY title
>28 STORED
<28 set page_content 0 0 26
> FOUND KEY page_content
>28 STORED
<28 set is_offline 0 0 5
> FOUND KEY is_offline
>28 STORED
<28 set allow_registrations 0 0 5
> FOUND KEY allow_registrations
>28 STORED
```

The key value pairs: title, page_content, is_offline and allow_registrations were updated after the key “token” was deleted.

View first 50 lines after the key “email” was deleted for the first time.

Command: sed -n '444,544p' memcache-stripped.log

```
root@attackdefense:~# sed -n '501,550p' memcache-stripped.log
<29 delete email
> FOUND KEY email
>29 DELETED
<28 get title
> FOUND KEY title
>28 sending key title
>28 END
<28 get page_content
> FOUND KEY page_content
>28 sending key page_content
>28 END
LRU crawler thread sleeping
<30 new auto-negotiating client connection
30: Client using the ascii protocol
```

```
<30 set sessid_11131211224 0 0 62
> FOUND KEY sessid_11131211224
>30 STORED
<28 get token
> FOUND KEY token
>28 sending key token
>28 END
<28 get allow_registrations
> FOUND KEY allow_registrations
>28 sending key allow_registrations
>28 END
<29 delete sessid_11131211224
> FOUND KEY sessid_11131211224
>29 DELETED
<28 get is_offline
> FOUND KEY is_offline
>28 sending key is_offline
>28 END
<28 get title
> FOUND KEY title
>28 sending key title
>28 END
<28 get page_content
> FOUND KEY page_content
>28 sending key page_content
>28 END
```

Only key-value pair, sessid_11131211224 was updated after the key “email” was deleted.

View first 50 lines after the key “token” was deleted for the second time.

Command: sed -n '541,591p' memcache-stripped.log

```
root@attackdefense:~# sed -n '541,591p' memcache-stripped.log
<29 delete token
> FOUND KEY token
>29 DELETED
<28 get token
> NOT FOUND token
>28 END
<28 set token 0 0 32
> NOT FOUND token
>28 STORED
<28 set title 0 0 30
> FOUND KEY title
>28 STORED
<28 set page_content 0 0 26
> FOUND KEY page_content
>28 STORED
<28 set is_offline 0 0 5
> FOUND KEY is_offline
>28 STORED
<28 set allow_registrations 0 0 5
> FOUND KEY allow_registrations
>28 STORED
```

The key value pairs: title, page_content, is_offline and allow_registrations were updated after the key “token” was deleted.

Q5. In order to leverage the compromised Memcached server to perform DDoS attack, the attacker had updated one of the key-value pairs stored on the Memcached server with a very large string. Identify the key-value pair which was updated.

Answer: title

Solution:

Step 1: Filter the commands which can be executed to update a key value pair and check for the content length of each updated key value pair.

Command: cat memcache-stripped.log | grep -v "<28" |grep -E 'set |replace |append |prepend '

```
root@attackdefense:~# cat memcache-stripped.log | grep -v "<28" |grep -E 'set |replace |append |prepend '
<30 set is_offline 0 0 4
<30 set sessid_11131211224 0 0 62
<29 replace password 0 3600 5
<30 set title 0 0 200000
root@attackdefense:~#
```

Step 2: Check the content length of title key value pair before it was modified (i.e. log entries before line 579).

Commands:
cat memcache-stripped.log | grep -v "<28" |grep -nE 'set title'
head -578 memcache-stripped.log | grep 'set title'

```
root@attackdefense:~# cat memcache-stripped.log | grep -v "<28" |grep -nE 'set title'
579:<30 set title 0 0 200000
root@attackdefense:~#
root@attackdefense:~# head -578 memcache-stripped.log | grep 'set title'
<28 set title 0 0 30
<28 set title 0 0 30
<28 set title 0 0 30
<28 set title 0 0 30
root@attackdefense:~#
```

The initial content length of the value stored in key “title” was 30, but the value of key “title” was updated and the final content length was 200,000

References:

1. Memcached (https://memcached.org/)