time_to_attack_n

May 31, 2023

[1]:
```
# basic parameters per cluster
nservers = 412
server_memory = (196 - 10)*10**9 # 96 GB
ncores_per_server = 40
hashes_sec_core = 2**25.015134275003597
dict_queries_sec = 2**20.863350 # we've slightly a better number
t_sec =  1 * 24 * 3600
nhashes_stored = 2**60
hashes_sec_phase_i = 2**24.72
dict_add_sec = 2**23.41
log2_l = 52 # log2 of #hashes stored
nhashes_stored = 2**log2_l
```

[2]:
```
# basic functions

def seconds_2_time(t):
    """
    Convert seconds into understandable string
    """

    from math import floor
    t = float(t)
    days  = floor(t/(3600*24))
    t = t - days*24*3600
    hours = floor(t/3600)
    t = t - hours*3600
    minutes = floor(t/60)
    t = t - minutes*60

    return f"{days} days, {hours} hours, {minutes} mins, {floor(t)} sec"



def nqueries_sender(nsenders, hashes_sec_core, difficulty=0):
    """ Return how many queries senders can generate per second """

    return nsenders*hashes_sec_core/(2**difficulty)
```

```
def nqueries_receiver(nreceivers, dict_queries_sec):
    """
    Return how many queries receivers can make in a second
    """
    return nreceivers * dict_queries_sec

# phase 0 of phase ii
def t_regen_msg(nsenders,
                nreceivers,
                hashes_sec_core,
                dict_add_sec,
                difficulty,
                nhashes_stored):
    """
    return number of seconds needed to regenerate the long message
    """
    nsecs_sender = nhashes_stored / nqueries_sender(nsenders,
                                                    hashes_sec_core,
                                                    difficulty=0)

    # how many hashes receivers will try to store?
    nhashes_receiver = (nhashes_stored/(2**difficulty))
    nsecs_receiver = nhashes_receiver / nqueries_receiver(nreceivers,
                                                          dict_add_sec)


    # since we will wait for everyone to finish
    return max(nsecs_receiver, nsecs_sender)


# phase 1 of phase ii
def t_enough_candidates(n,
                        nsenders,
                        nreceivers,
                        hashes_sec_core,
                        dict_add_sec,
                        difficulty,
                        nhashes_in_dict,
                        verbose=False):
    """
    Return time needed to generate enough candidates
    """
    from math import log2

    # how many hashes sender has to make
    nreq_qsender = (2**n/nhashes_in_dict)
```

```
    # time needed for senders to get this number of queries
    t_req_qsender = nreq_qsender / nqueries_sender(nsenders,
                                                   hashes_sec_core,
                                                        difficulty)
    # how many queries reciever needs to get enough candidates?
    # it should be less than or equal than number of hashes
    nreq_qrecv = nreq_qsender/(2**difficulty)
    t_req_qrecv = nreq_qrecv/nqueries_receiver(nreceivers,
                                                   dict_queries_sec)

    if (verbose):
        print(f"t2: nsndr={nsenders}, t_nsndrs={int(t_req_qsender)/3600.0:.
↪2f}h, nrcvs={nreceivers}, t_rcvs={int(t_req_qrecv)/3600.0:.2f}h,␣
↪diff={difficulty}")

    return max(t_req_qrecv, t_req_qsender)
```

[3]:
```
def lg2_ncnd(n,
             dict_elm_size,
             server_memory,
             nservers,
             ncores_per_server,
             nreceivers,
             difficulty):

    """
    Return log2(ncandidates) that will be stored during phase_ii
    Most of them are false positives.
    This will be used to estimate the complexity of phase_iii
    """

    from math import log2, ceil, floor

    nreceivers_per_server = nreceivers // nservers
    nslots_priv =  (server_memory/nreceivers_per_server) / dict_elm_size
    nbucket = 512 / (8*dict_elm_size)

    index_size_bits = log2(nslots_priv/nbucket)
    index_size_bytes = ceil(index_size_bits/8)

    n_used_bits = floor(log2(nservers))\
                + dict_elm_size*8\
                + index_size_bits\
                + difficulty\
                + log2(nreceivers*nservers)
```

```
    return ceil(n - n_used_bits)
```

[4]:
```
def best_parameter(n,
                   nservers,
                   server_memory,
                   ncores_per_server,
                   nhashes_stored,
                   hashes_sec_core,
                   dict_queries_sec,
                   dict_elm_size=4,
                   verbose=False):

    """
    Find the best parameters: nsenders, nreceivers, and difficulty
    that minimizes the run time on given cluster.
    """
    from math import log2
    # step 1 loop over decompose nsenders, nreceivers
    # step 2  loop over difficulty
    # step 3 find the time
    # step 4 store the minimum
    t_min = float("inf")
    # how many hashes servers can store
    max_nhashes_in_memory = server_memory*nservers / dict_elm_size
    nhashes_in_dict = min(max_nhashes_in_memory, nhashes_stored)

    if (verbose):
        print(f"Memory can take at most 2^{log2(max_nhashes_in_memory)} hashes")
        print("++++++++++++++++++++++++++++++++++++++++++++++++++\n\n")

    for nreceivers in range(nservers,
                            nservers*ncores_per_server,
                            nservers):

        nsenders = nservers*ncores_per_server - nreceivers

        for difficulty in range(20):
            t1 = t_regen_msg(nsenders,
                             nreceivers,
                             hashes_sec_core,
                             dict_add_sec,
                             difficulty,
                             nhashes_stored)

            if (t1<0):
                continue
```

```
            #nhashes_in_dict = nhashes_stored


            t2 = t_enough_candidates(n,
                                     nsenders,
                                     nreceivers,
                                     hashes_sec_core,
                                     dict_add_sec,
                                     difficulty,
                                     nhashes_in_dict,
                                     verbose=verbose)

            t = t1 + t2
            if (t < t_min):
                if (verbose):
                    print(f"t={seconds_2_time(t)}\n"
                          +f"t1={seconds_2_time(t1)}\n"
                          +f"t2={seconds_2_time(t2)}\n"
                          +f"cpu_hours={(t // 3600)*nservers*ncores_per_server}"
                          +f"nsednders={nsenders}, nreceivers={nreceivers}, "
                          +f"difficulty={difficulty}\n"
                          +f"log2(nhashes)={log2(nhashes_in_dict)}")
                    print("================================\n\n")

                t_min = t
                t1_min = t1
                t2_min = t2
                nsenders_min = nsenders
                nreceivers_min = nreceivers
                difficulty_min = difficulty
                nhashes_in_dict_min = nhashes_in_dict

    return {"t" : seconds_2_time(t_min),
            "t1" : seconds_2_time(t1_min),
            "t2" : seconds_2_time(t2_min),
            "cpu_hours": (t_min // 3600)*nservers*ncores_per_server,
            "nsenders" : nsenders_min,
            "nreceivers" : nreceivers_min,
            "difficulty" : difficulty_min,
            "lg2(nhashes_in_dict)" : log2(nhashes_in_dict_min)}
```

[5]:
```
n = 96
elm_size = 4 # u32
chx = best_parameter(n,
                nservers, # 412
                server_memory, # 192GB
```

```
                ncores_per_server, # 40 cores
                nhashes_stored, # 2^52
                hashes_sec_core,
                dict_queries_sec,
                dict_elm_size=elm_size,
                verbose=False)

# how many candidate (including false positive)
j = lg2_ncnd(n,
        elm_size,
        server_memory,
        nservers,
        ncores_per_server,
        chx['nreceivers'],
        chx['difficulty'])

print(f"total time  = {chx['t']}\n"
    + f"rgen msg    = {chx['t1']}\n"
    + f"prob & hash = {chx['t2']}\n"
    + f"cpu_hours  = {chx['cpu_hours']}h\n"
    + f"nsenders   = {chx['nsenders']}\n"
    + f"receivers  = {chx['nreceivers']}\n"
    + f"difficulty = {chx['difficulty']}\n"
    + f"lg2(nhashes_in_dict) = {chx['lg2(nhashes_in_dict)']}")

print("----------------------")

print(f"est. n of stored candidates  = 2^{j}")
```

```
total time  = 0 days, 22 hours, 6 mins, 47 sec
rgen msg    = 0 days, 4 hours, 42 mins, 46 sec
prob & hash = 0 days, 17 hours, 24 mins, 1 sec
cpu_hours  = 362560.0h
nsenders   = 7828
receivers  = 8652
difficulty = 2
lg2(nhashes_in_dict) = 44.123012192277514
----------------------
est. n of stored candidates  = 2^6
```

[6]:
```
n = 96
elm_size = 2 # u16
chx = best_parameter(n,
                nservers, # 412
                server_memory, # 192GB
                ncores_per_server, # 40 cores
                nhashes_stored, # 2^52
```

```
                hashes_sec_core,
                dict_queries_sec,
                dict_elm_size=elm_size,
                verbose=False)

# how many candidate (including false positive)
j = lg2_ncnd(n,
        elm_size,
        server_memory,
        nservers,
        ncores_per_server,
        chx['nreceivers'],
        chx['difficulty'])

print(f"total time  = {chx['t']}\n"
    + f"rgen msg    = {chx['t1']}\n"
    + f"prob & hash = {chx['t2']}\n"
    + f"cpu_hours  = {chx['cpu_hours']}h\n"
    + f"nsenders   = {chx['nsenders']}\n"
    + f"receivers  = {chx['nreceivers']}\n"
    + f"difficulty = {chx['difficulty']}\n"
    + f"lg2(nhashes_in_dict) = {chx['lg2(nhashes_in_dict)']}")

print("---------------------")

print(f"est. n of stored candidates  = 2^{j}")
```

```
total time  = 0 days, 13 hours, 24 mins, 47 sec
rgen msg    = 0 days, 4 hours, 42 mins, 46 sec
prob & hash = 0 days, 8 hours, 42 mins, 0 sec
cpu_hours  = 214240.0h
nsenders   = 7828
receivers  = 8652
difficulty = 2
lg2(nhashes_in_dict) = 45.123012192277514
---------------------
est. n of stored candidates  = 2^22
```

[7]:
```
n = 96
elm_size = 1 # u8
chx = best_parameter(n,
                nservers, # 412
                server_memory, # 192GB
                ncores_per_server, # 40 cores
                nhashes_stored, # 2^52
                hashes_sec_core,
                dict_queries_sec,
```

```
                dict_elm_size=elm_size,
                verbose=False)

# how many candidate (including false positive)
j = lg2_ncnd(n,
        elm_size,
        server_memory,
        nservers,
        ncores_per_server,
        chx['nreceivers'],
        chx['difficulty'])

print(f"total time  = {chx['t']}\n"
    + f"rgen msg    = {chx['t1']}\n"
    + f"prob & hash = {chx['t2']}\n"
    + f"cpu_hours  = {chx['cpu_hours']}h\n"
    + f"nsenders   = {chx['nsenders']}\n"
    + f"receivers  = {chx['nreceivers']}\n"
    + f"difficulty = {chx['difficulty']}\n"
    + f"lg2(nhashes_in_dict) = {chx['lg2(nhashes_in_dict)']}")

print("----------------------")

print(f"est. n of stored candidates  = 2^{j}")
```

```
total time  = 0 days, 8 hours, 46 mins, 0 sec
rgen msg    = 0 days, 3 hours, 5 mins, 44 sec
prob & hash = 0 days, 5 hours, 40 mins, 15 sec
cpu_hours  = 131840.0h
nsenders   = 11948
receivers  = 4532
difficulty = 3
lg2(nhashes_in_dict) = 46.123012192277514
----------------------
est. n of stored candidates  = 2^29
```