CS 370
Introduction to Security

Web Security Basics and Password

Oregon State University

The Internet w/o SSL/TLS

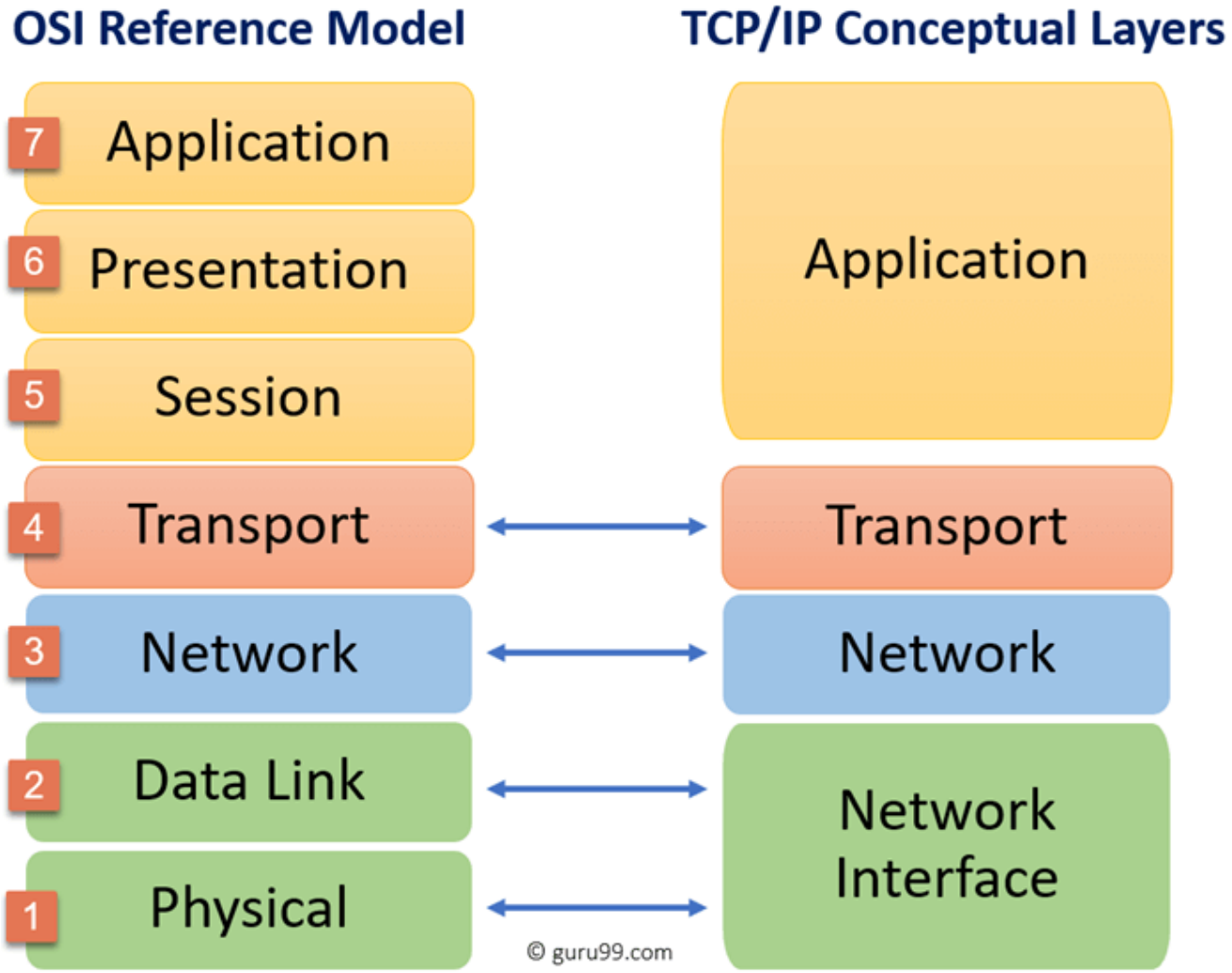
Everybody in the middle knows that I searched ‘dogs’ and they also know the search result…

Why Transport Layer Security?

- OSI vs IP Layers

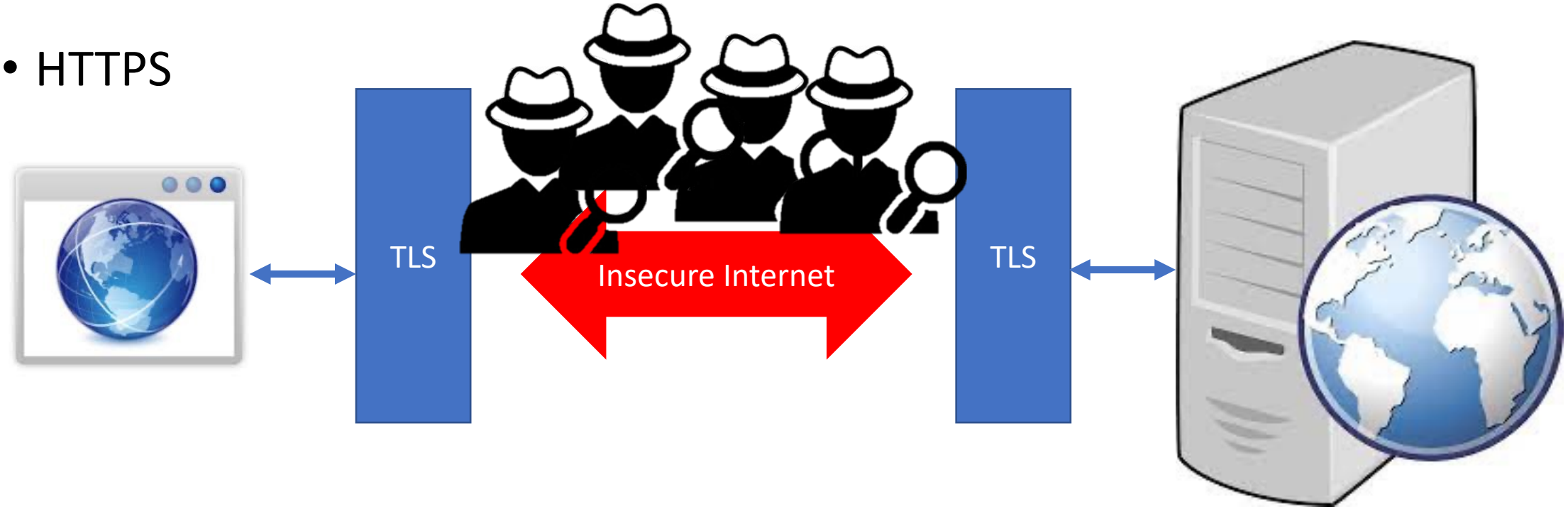
- TLS would like to be
  - Transparent to applications

If transparent,
Applications does not have to change their behavior for adopting security to the network protocols

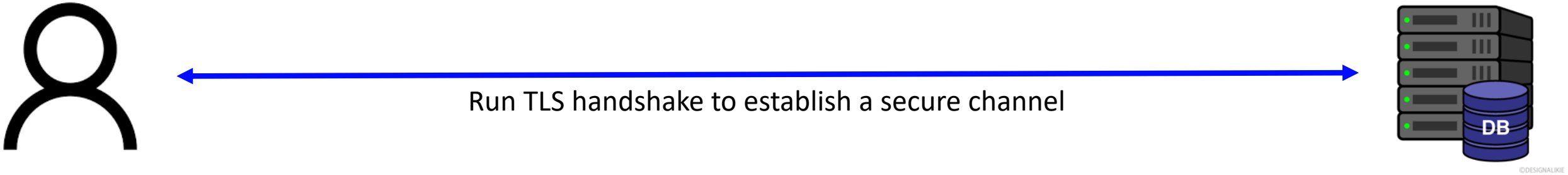


A Web Server Example

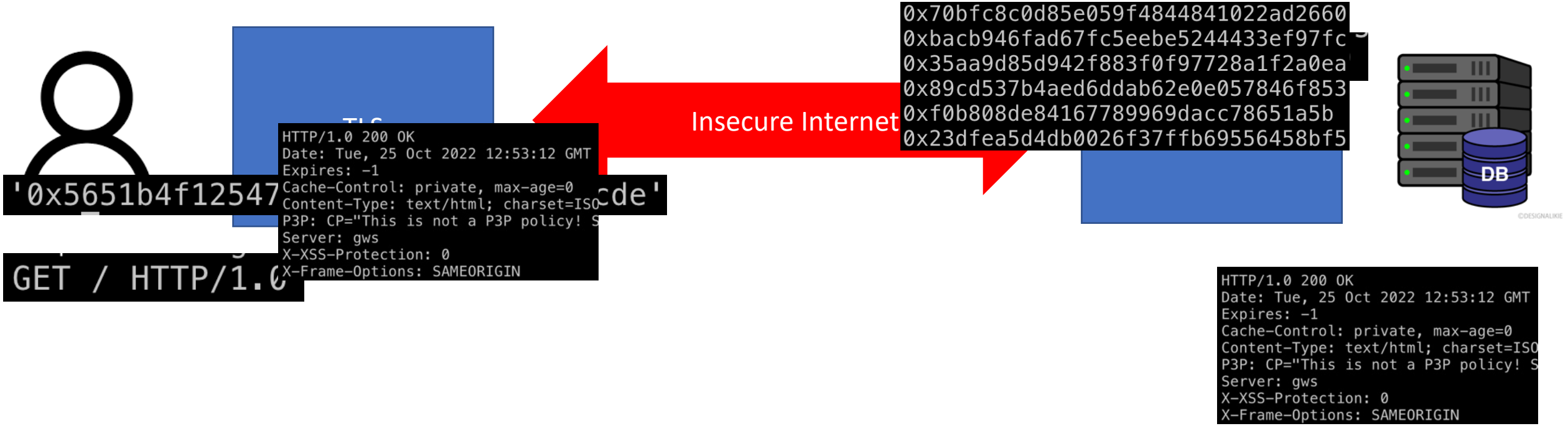
- HTTPS



A Web Server Example



A Web Server Example



HTTP/1.0 200 OK
Date: Tue, 25 Oct 2022 12:53:12 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO
P3P: CP="This is not a P3P policy! S
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN

SSL/TLS Steps

Client

- 1. Client hello

Server

- 2. Server hello
- 3. Server Certificate
- 4. Server Key Exchange
- 5. Server Hello Done

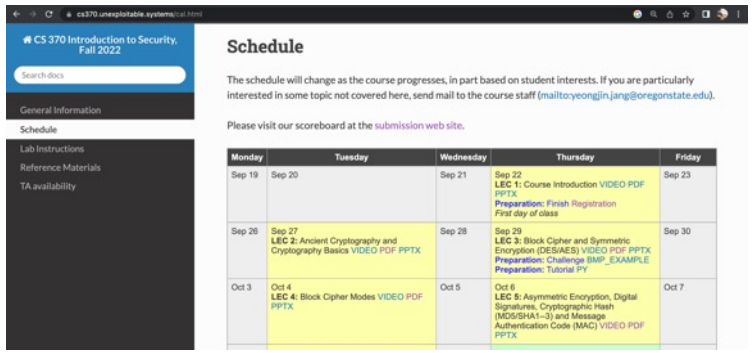
- 6. Client Key Exchange
  - Shares DH material after verifying server signature for server's DH material
- 7. Change Cipher Spec
- 8. Encrypted Handshake Message

- 9. Change Cipher Spec
- 10. Encrypted Handshake Message

World Wide Web

- A way of accessing documents (web pages) over the internet
  - So-called the Internet for non-techie folks

- Uses HTTP as a document-delivery protocol
  - Request: GET /index.html HTTP/1.0\r\n
  - Response: 200 OK HTTP/1.0\r\n
  - … contents …



How Can We Apply Access Control On Web?

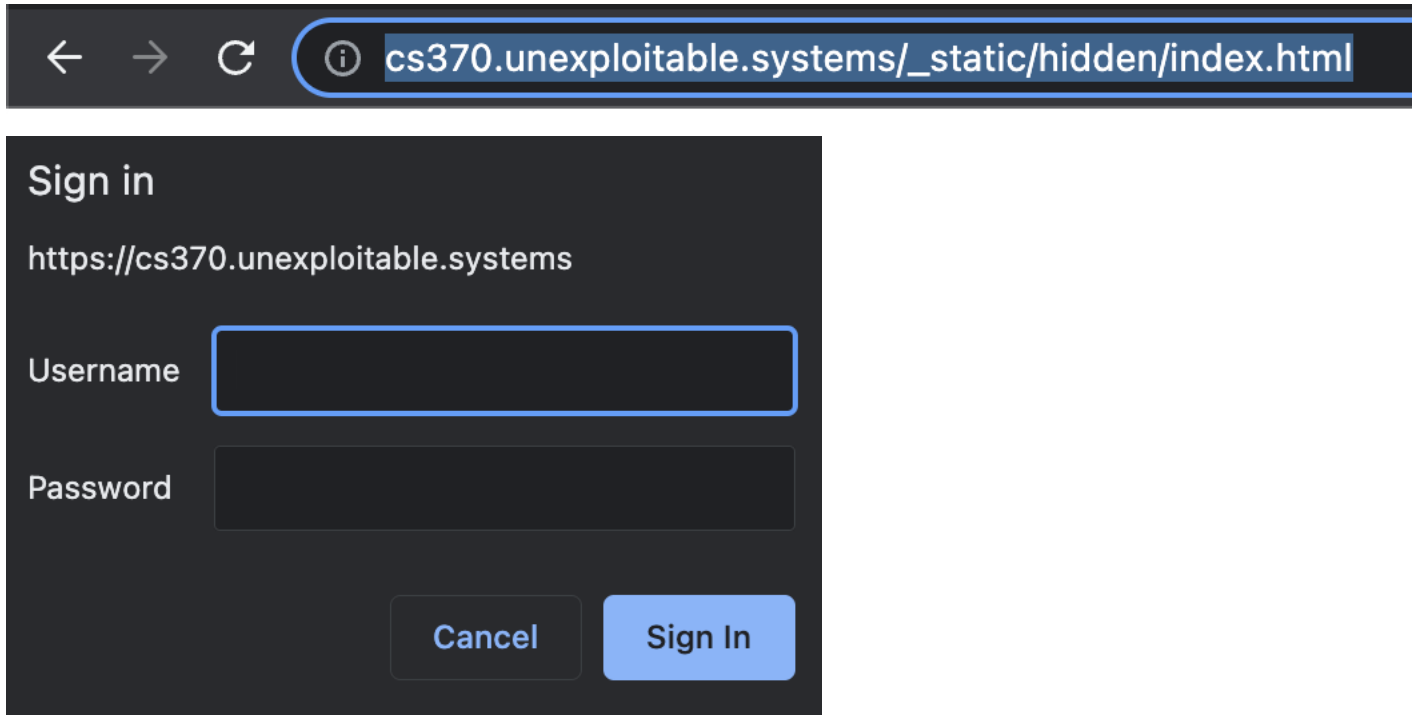
- Without the protection, anyone can access the webpage via URL
  - URL: Uniform Resource Locator
  - http://cs370.unexploitable.systems

- Can we apply the access control on the website?
  - Preferably with a password?

- E.g., when accessing the website, type
  - ID: blue9057
  - Password: some password…

HTTP Basic Authentication

- HTTP has a ‘Basic’ authentication method..
  - https://cs370.unexploitable.systems/_static/hidden/index.html



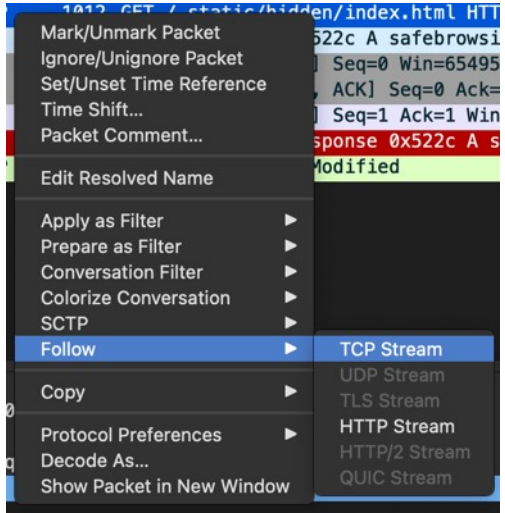
HTTP Basic Auth: Insecure

HTTP is **unencrypted;**
All the **middlemen can see** your authorization data..

- HTTP – packets are in plaintext
  - https://cs370.unexploitable.systems/_static/http_basic.pcapng

1	0.000000000	127.0.0.1	127.0.0.1	HTTP	1012	GET /_static/hidden/index.html HTTP/1.1
2	0.001963698	127.0.0.1	127.0.0.53	DNS	83	Standard query 0x522c A safebrowsing.google.com
3	0.002251748	127.0.0.1	127.0.0.1	TCP	74	53732 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1993954412 TSec…
4	0.002275826	127.0.0.1	127.0.0.1	TCP	74	8080 → 53732 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1993…
5	0.002306468	127.0.0.1	127.0.0.1	TCP	66	53732 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1993954413 TSecr=1993954413
6	0.017663091	127.0.0.53	127.0.0.1	DNS	118	Standard query response 0x522c A safebrowsing.google.com CNAME sb.l.google.com A 142…
7	0.025120028	127.0.0.1	127.0.0.1	HTTP	254	HTTP/1.1 304 Not Modified

- Stream:




```
GET /_static/hidden/index.html HTTP/1.1
Host: cs370.unexploitable.systems:8080
Connection: keep-alive
Cache-Control: max-age=0
Authorization: Basic Ymx1ZTkwNTc6Y3MzNzB7QjRzSWNfQXVUaF9JNV9OMHRfczNDdVIzfQ==
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0
Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/
*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: _jsuid=1158429791;
experimentation_subject_id=eyJfcmFpbHMiOnsibWVzc2FnZSI6Iklqa3paak01WVdZekxUYzBOV0V0TkdJMU5pMDVZelpsTFRVd
05HUm1abVExTlRKak9DST0iLCJleHAiOm51bGwsInB1ciI6ImNvb2tpZS5leHBlcmltZW50YXRpb25fc3ViamVjdF9pZCJ9fQ%3D%3D-
-14df51e13094f383b80e4b21ff0c195dd82560ed; _jsuid=1158429791
If-None-Match: W/"6360b363-25"
If-Modified-Since: Tue, 01 Nov 2022 05:49:23 GMT

HTTP/1.1 304 Not Modified
Server: nginx/1.14.0 (Ubuntu)
Date: Tue, 01 Nov 2022 06:01:09 GMT
Last-Modified: Tue, 01 Nov 2022 05:49:23 GMT
Connection: keep-alive
ETag: "6360b363-25"
```

What is that string??

- Base64 Encoding
  - Binary to Text encoding
  - Uses printable 64-characters
- Suppose you have a string “ASD”
- 01000001 01010011 01000100
- 010000 010101 001101 000100 (6 bits)
- Q V N E

```
>>> base64.b64encode(b"ASD")
b'QVNE'
```

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

Base64 Can Encode Binary to String!

- Base64 Encoding
  - Binary to Text encoding
  - Uses printable 64-characters
- Suppose you have a string “ffe0e8” (hex)
- 11111111 11100000 11101000
- 111111 111110 000011 101000 (6 bits)
- / + D o

```
>>> base64.b64encode(b"\xff\xe0\xe8")
b'/+Do'
```

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

Characteristics of a Base64 String

- Base64 Encoding
  - All printable characters
  - Has / and + in addition to
  - [A-Za-z0-9]

- Ymx1/ZTkwNTc6Y3MzNzB7+QjRzSWNfQXVUaF9JNV9OMHRfczNDdVIzfQ==
- https://www.base64decode.net/

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

Let's Decode the String

```
GET /_static/hidden/index.html HTTP/1.1
Host: cs370.unexploitable.systems:8080
Connection: keep-alive
Cache-Control: max-age=0
Authorization: Basic Ymx1ZTkwNTc6Y3MzNzB7QjRzSWNfQXVUaF9JNV9OMHRfczNDdVIzfQ==
```

Ymx1ZTkwNTc6Y3MzNzB7QjRzSWNfQXVUaF9JNV9OMHRfczNDdVIzfQ==

blue9057:cs370{B4sIc_AuTh_I5_N0t_s3CuR3}

Username: blue9057
Password: cs370{B4sIc_AuTh_I5_N0t_s3CuR3}

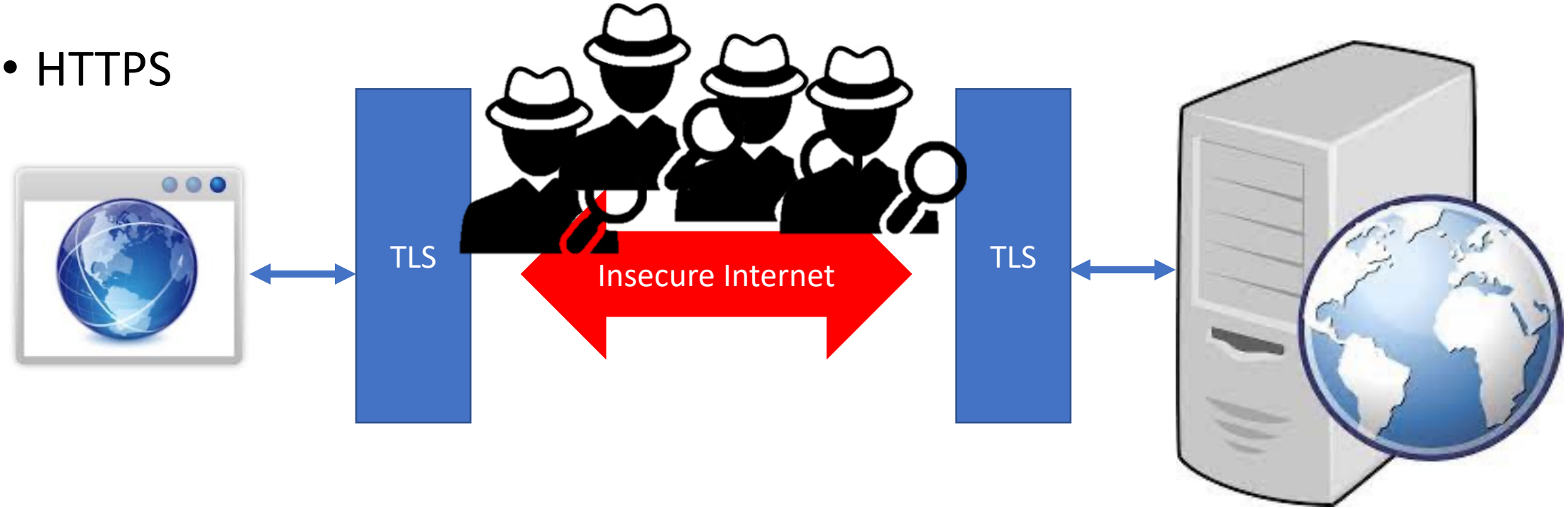
Implications

- We can apply HTTP basic authentication to apply
  - Access control on a webpage
  - Users are required to type the matching username and password
  - Otherwise, you can’t access the page

- Problem
  - HTTP is unencrypted
  - base64Encode(username:password) is transmitted in
    - Every HTTP request…

Solution: Use HTTPS

- HTTPS



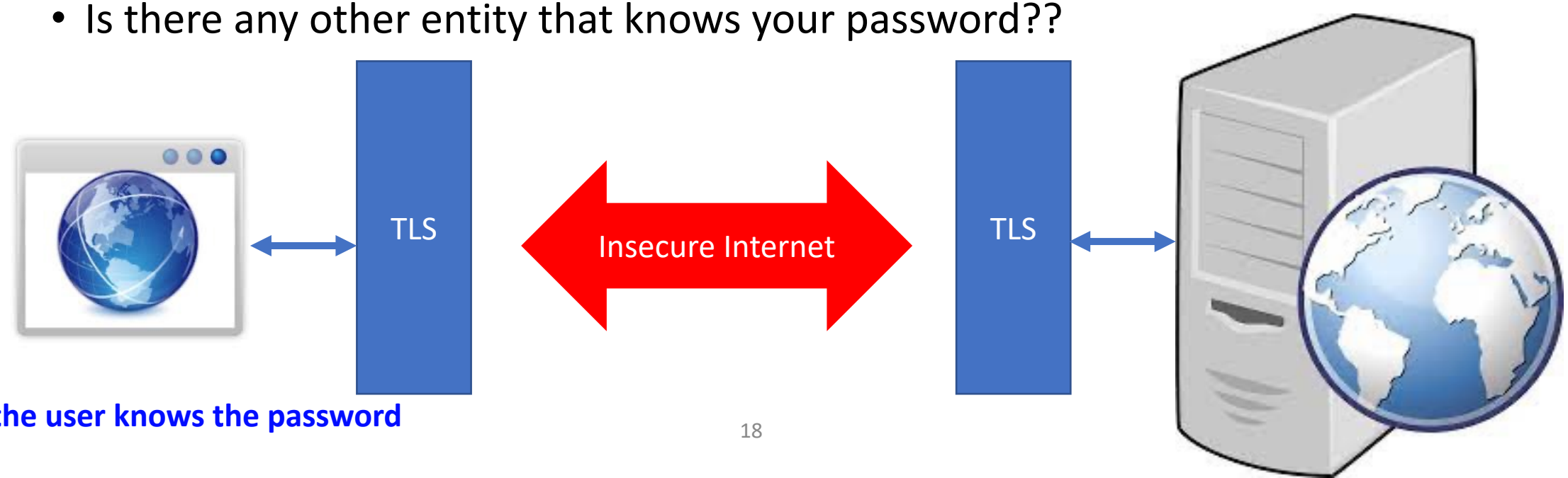
- No one other than the server/client can see the content!

Why Do We Use a Password?

- Password is the factor of authentication that
  - What you know

- What is the problem of this approach?
  - Is there any other entity that knows your password??

The server also knows your password



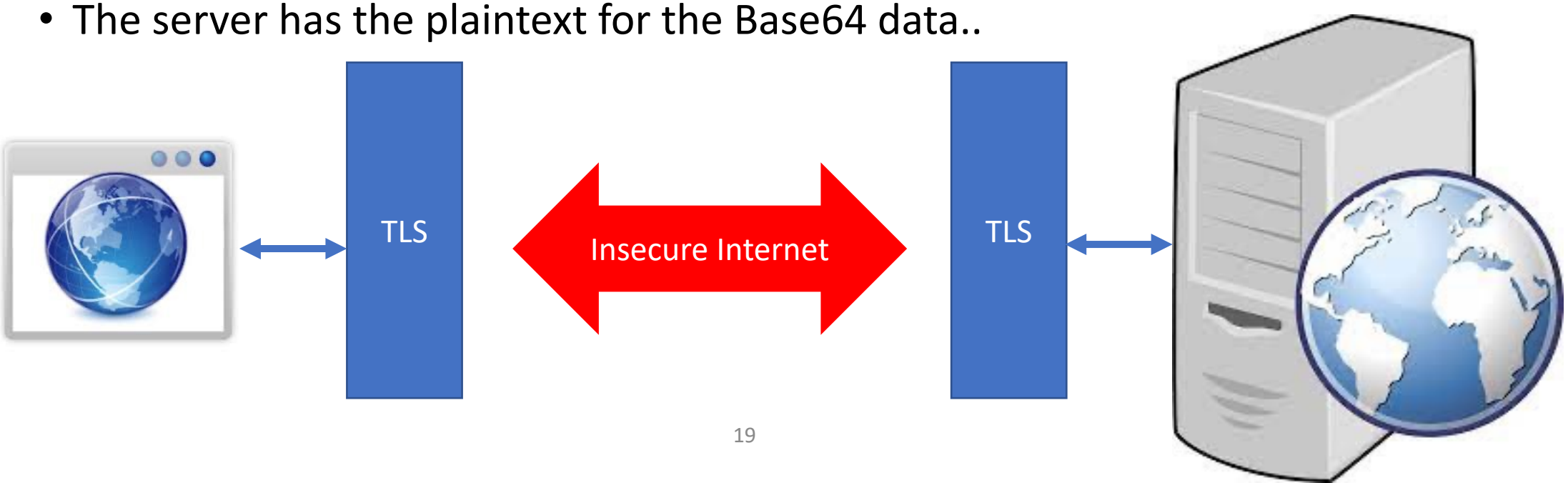
You as the user knows the password

Why is this even a problem?

- Password is set based on what 'only you know'
  - No one else should know the password
- HTTPS basic authentication
  - No one else knows your password but
  - The server has the plaintext for the Base64 data..

Ymx1ZTkwNTc6Y3MzNzB7QjRzSWNfQXVUaF9JN
V9OMHRfczNDdVIzfQ==

blue9057:cs370{B4sIc_AuTh_I5_N0t_s3CuR3}



Server stores the Password

- So the server knows and stores the password

Home › Email Security

Bed Bath & Beyond Investigating Data Breach After Employee Falls for Phishing Attack

By Eduard Kovacs on November 01, 2022

Share Tweet Recommend 0 RSS

Bed Bath & Beyond revealed last week in an SEC filing that it recently suffered a data breach after an employee fell victim to a phishing attack.

This is not the first time Bed Bath & Beyond has disclosed a cybersecurity incident. In 2019, the retailer revealed that some customer accounts had been breached. At the time, it said hackers had obtained username and password combinations from a breach at a different company and relied on the fact that many people use the same credentials for multiple online accounts.

Server stores the Password

- So the server knows and stores the password

Home › Email Security

Bed Bath & Beyond Investigating Data Breach After Employee Falls for Phishing Attack

By Eduard Kovacs on November 01, 2022

Share Tweet Recommen

Bed Bath & Beyond revealed last breach after an employee fell vic

This is not the first time Bed Bath & the retailer revealed that some cust hackers had obtained username and company and relied on the fact that many people use the same credentials for multiple online accounts.

Server can be **hacked**

Passwords stored in the server **could also be leaked**

How many passwords do you have, and do you **re-use** the same password in **multiple websites**??

Can we hide the password from the server??

- YES

- How?
  - Instead of storing the password directly, we store
  - SHA256(“some_secret” + password)

  - E.g.,
  - SHA256(“some_secret” + “my-super-secure-password!@#$11”)
  - 59636881ab9bf34263cf3f4d90f25d2b91e74e8804b802d25c8f4bc5c80846ee

Hashed-password

- 59636881ab9bf34263cf3f4d90f25d2b91e74e8804b802d25c8f4bc5c80846ee
  - SHA256(“some_secret” + “my-super-secure-password!@#$11”)

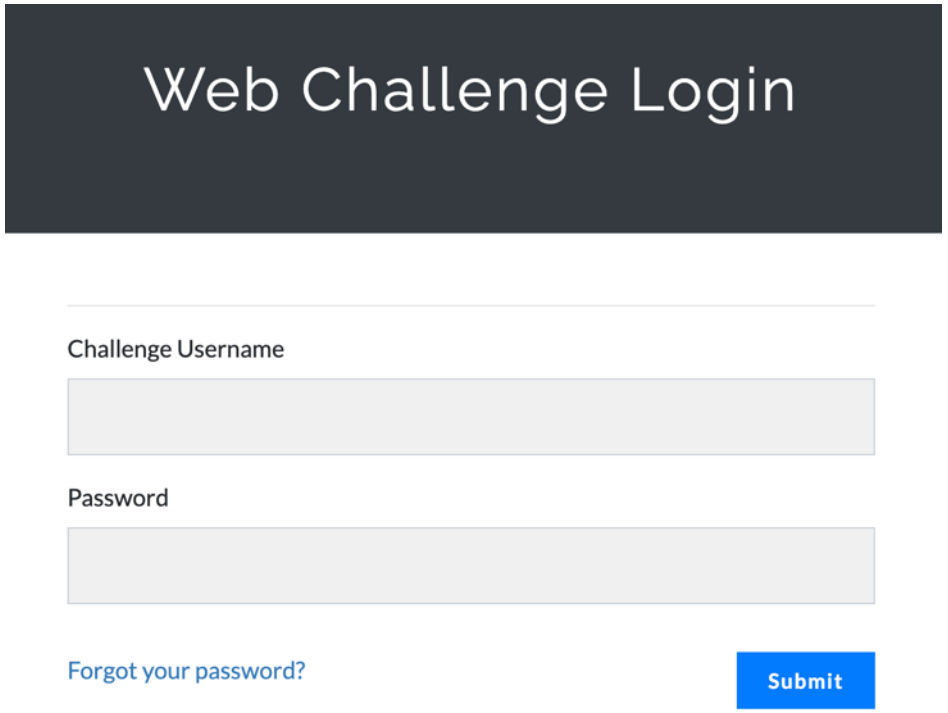
- Can the attackers who steal the hash recover the password?
  - This is the same as, what is the inverse of “59636881ab9bf34263cf3f4d90f25d2b91e74e8804b802d25c8f4bc5c80846ee”

  - And finding an inverse of SHA256 is technically infeasible…

Web Server Authentication

- We don’t use HTTP Basic Authentication for logging into the website

- Instead, we use the login form:
  - How does it work?



Web Password Authentication

- Send ID/password but the server stores hash of the password

blue9057: 59636881ab9bf34263cf3f4d90f25d2b91e74e8804b802d25c8f4bc5c80846ee

blue9057:my-super-secure-password!@#$11

Protected by TLS (HTTPS)

SHA256(“some_secret” + “my-super-secure-password!@#$11”)
= 59636881ab9bf34263cf3f4d90f25d2b91e74e8804b802d25c8f4bc5c80846ee

Server Logic

- User submits ID and password

- Query database to find a pair of
  - (Username, SHA256(secret + password))
  - Exists…

Database Access for Password

- SQL Query (SQL examples: https://www.w3schools.com/sql/sql_examples.asp)

  SELECT (username, password) FROM users WHERE username = ‘blue9057’
  and password = SHA256(secret + “my-super-secure-password!@#$11”)

- The user sends the plaintext password to the server
- But the password is never stored in the DB

- Even if the attackers know the secret, they have to inverse SHA256
  - To get the plaintext password

Caveat: SHA256 is susceptible to the Brute-force Attacks

- Suppose an attacker breached into the server
  - They can have the database (all the password hashes)
  - They can have the program source code (the secret that generates hash…)

- They can run the following algorithm
  - for strings in all_strings:
    - If SHA256(secret + strings) == ‘59636881ab9bf34263cf3f4d90f25d2b91e74e8804b802d25c8f4bc5c80846ee’
      - print(strings)

Caveat: SHA256 is susceptible to the Brute-force Attacks

- They can run the following algorithm
  - for strings in all_strings:
    - If SHA256(secret + strings) == ‘59636881ab9bf34263cf3f4d90f25d2b91e74e8804b802d25c8f4bc5c80846ee’
      - print(strings)

- How long should it take (theoretically) ?
  - ~2 ** 256 seconds (seems forever)

- Not true

Humans choose password not Randomly...

- The security factor of SHA256 works correctly (2 ** 256) if
  - The input was chosen completely randomly

- How human users choose password?
  - Easy to memorize and type (12345678, password)
  - Some familiar phrases (orangebeaverROCKS)
  - Add some numbers on it (password1234)
  - Add some special characters on in (password1234!!)

Top 30 Most Used Passwords in the World

1	123456	11	abc123	21	princess
2	password	12	1234	22	letmein
3	123456789	13	password1	23	654321
4	12345	14	iloveyou	24	monkey
5	12345678	15	1q2w3e4r	25	27653
6	qwerty	16	000000	26	1qaz2wsx
7	1234567	17	qwerty123	27	123321
8	111111	18	zaq12wsx	28	qwertyuiop
9	1234567890	19	dragon	29	superman
10	123123	20	sunshine	30	asdfghjkl

Dictionary Attack

- Hackers may try popular, known password phrases first..

- https://github.com/danielmiessler/SecLists/tree/master/Passwords/Common-Credentials

```
ginger
princess
joshua
cheese
amanda
summer
love
ashley
6969
nicole
chelsea
biteme
matthew
access
yankees
987654321
dallas
austin
thunder
taylor
matrix
minecraft
```

Why Do We Need Special Chars in Password?

- [A-Z] * 8
  - 26 ** 8 = 208,827,064,576
- [A-Za-z] * 8
  - 52 ** 8 = 53,459,728,531,456
- [A-Za-z0-9] and special chars * 8
  - 95 ** 8 = 6,634,204,312,890,625

Hash Crackers

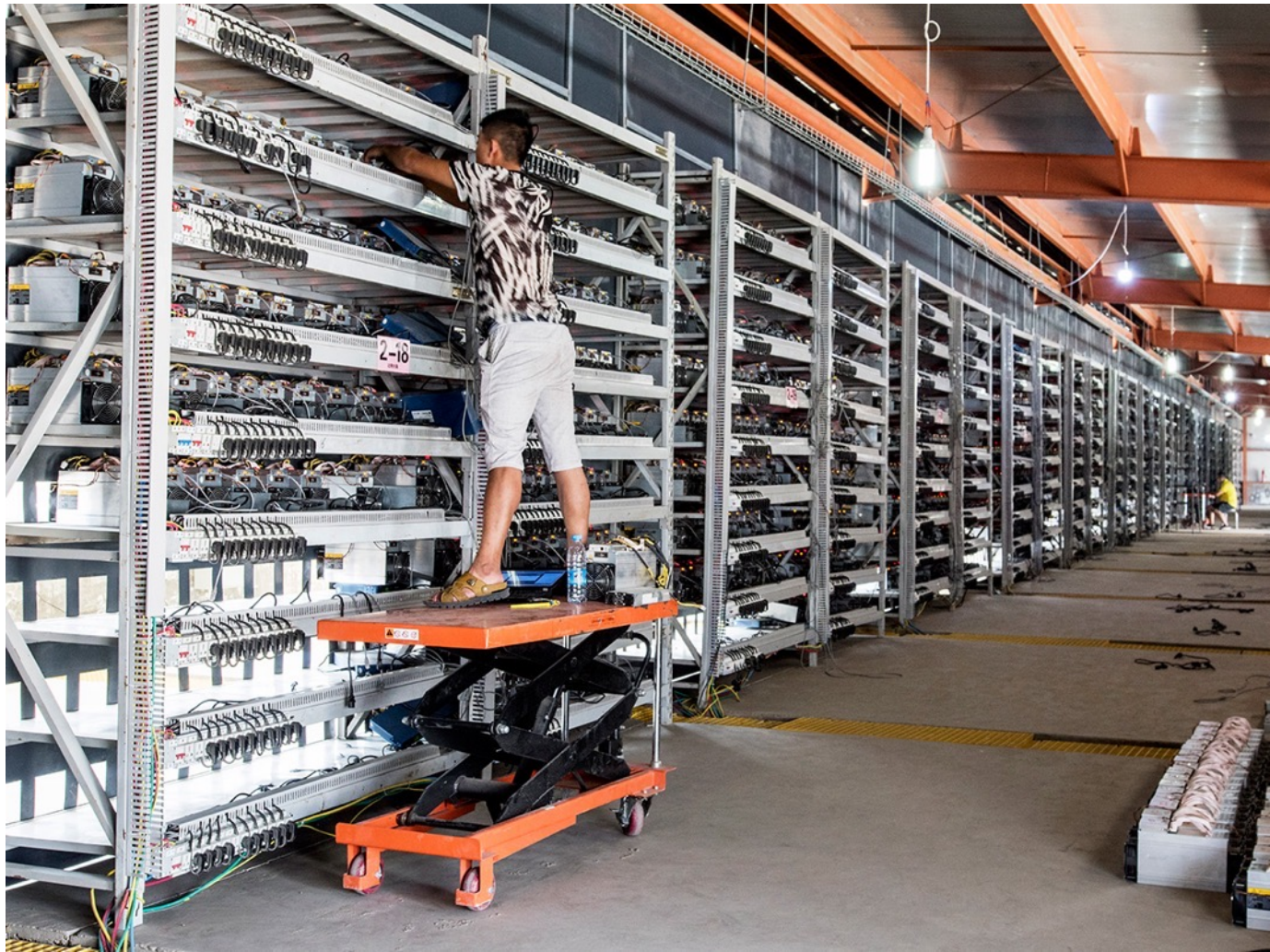
- We can run SHA256 with secret and try those passwords
- [A-Za-z0-9] and special chars * 8
  - 95 ** 8 = 6,634,204,312,890,625
- 6,634 trillion cases

Bitcoin Miners == Hash Crackers

- Using GPUs to calculate
  - X = transaction_data + random
  - SHA256(X)
  - The result should have 80 0s…

- 00000000000.....23948329793

1	Foundry USA	26.11 %	71.41 EH/s
2	AntPool	20.80 %	56.88 EH/s
3	F2Pool	14.82 %	40.54 EH/s
4	Binance Pool	9.73 %	26.63 EH/s
5	ViaBTC	9.29 %	25.42 EH/s



Bitcoin Miners == Hash Crackers

- [A-Za-z0-9] and special chars * 8
  - 95 ** 8 = 6,634,204,312,890,625
- 6,634 trillion cases
- 6P (Peta) cases (less than 1 ms???)

1	Foundry USA	26.11 %	71.41 EH/s
2	AntPool	20.80 %	56.88 EH/s
3	F2Pool	14.82 %	40.54 EH/s
4	Binance Pool	9.73 %	26.63 EH/s
5	ViaBTC	9.29 %	25.42 EH/s

SQL injection

- SQL Query (SQL examples: https://www.w3schools.com/sql/sql_examples.asp)

  SELECT (username, password) FROM users WHERE username = ‘blue9057’
  and password = ’my-super-secure-password!@#$11’

- What if we supply ‘or ‘a’=‘a as a password?

  SELECT (username, password) FROM users WHERE username = ‘blue9057’
  and password = ‘’ or ‘a’ =‘a’

  SELECT (username, password) FROM users WHERE
  username = ‘blue9057’ and password = ‘’ or
  ‘a’ =‘a’
  ALWAYS TRUE!!!

SQL injection

- What if we supply ‘or ‘a’=‘a as a password?

  SELECT (username, password) FROM users WHERE username = ‘blue9057’

  and password = ‘’ or ‘a’ =‘a’

- We can bypass password checking logic by injecting malicious data to the database SQL query

SQL injection

- What if we supply ’ union select (‘admin’, ‘a’) where ‘a’=‘a as a password?

  SELECT (username, password) FROM users WHERE username = ‘blue9057’

  and password = ‘’ union select (‘admin’, ‘a’) where ‘a’=‘a’

- It will fetch none for the first select and
- 2nd select will return will have
  - Username = ‘admin’
  - Password = ‘a’