

A3:2017 – Sensitive Data Exposure

Outline

- Real and hypothetical scenarios
- Rainbow Table
- Sensitive Data Exposure
 - Attack Vectors
 - Security Weakness
 - Impact
- Testing Sensitive Data Exposure
- Insecure Communication Exercise (WebGoat)
- Countermeasures



Tavis Ormandy 

@tavis0

Follow



Could someone from cloudflare security urgently contact me.

7:11 PM - 17 Feb 2017

243 Retweets **902** Likes



39



243



902

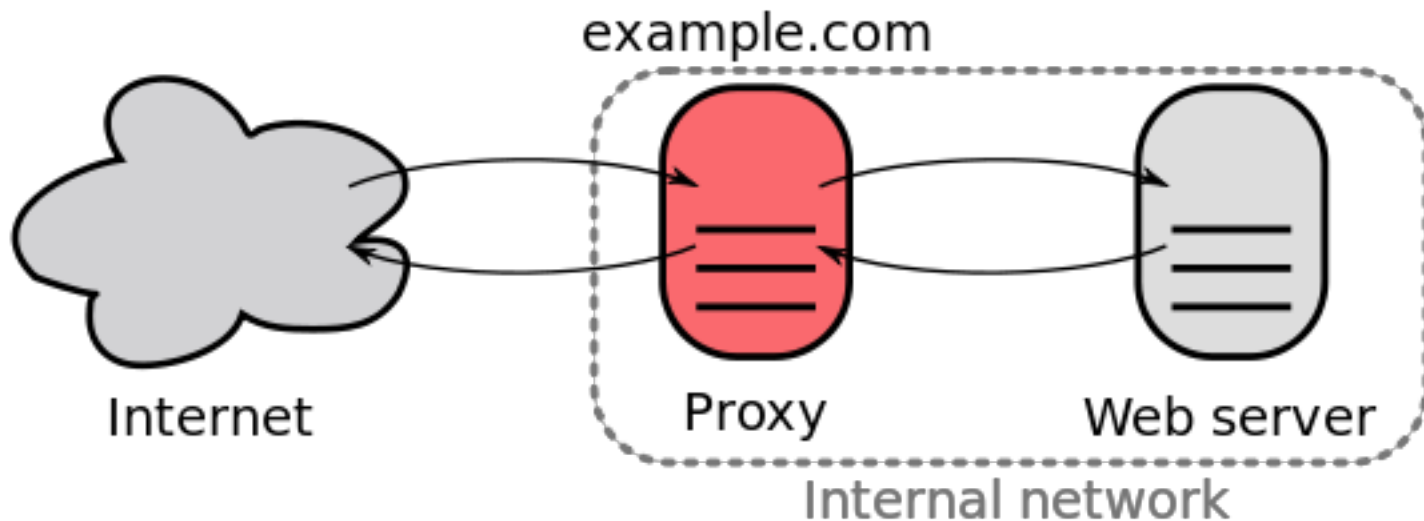


Cloudbleed

A security bug affecting Cloudflare's reverse proxies

Cloudbleed

A security bug affecting Cloudflare's **reverse proxies**



- Hide the origin of the server
- Can protect against DDoS attacks
- Can enforce TLS encryption if that is not applied at the server side

Cloudbleed

A security bug affecting Cloudflare's reverse proxies

- Discovered during fuzzing.
 - The way the broken html tags were parsed, caused information from a random portion of the server's memory to be returned.
- Leaked information:
 - HTTP cookies
 - authentication tokens
 - HTTP POST bodies and other sensitive data.

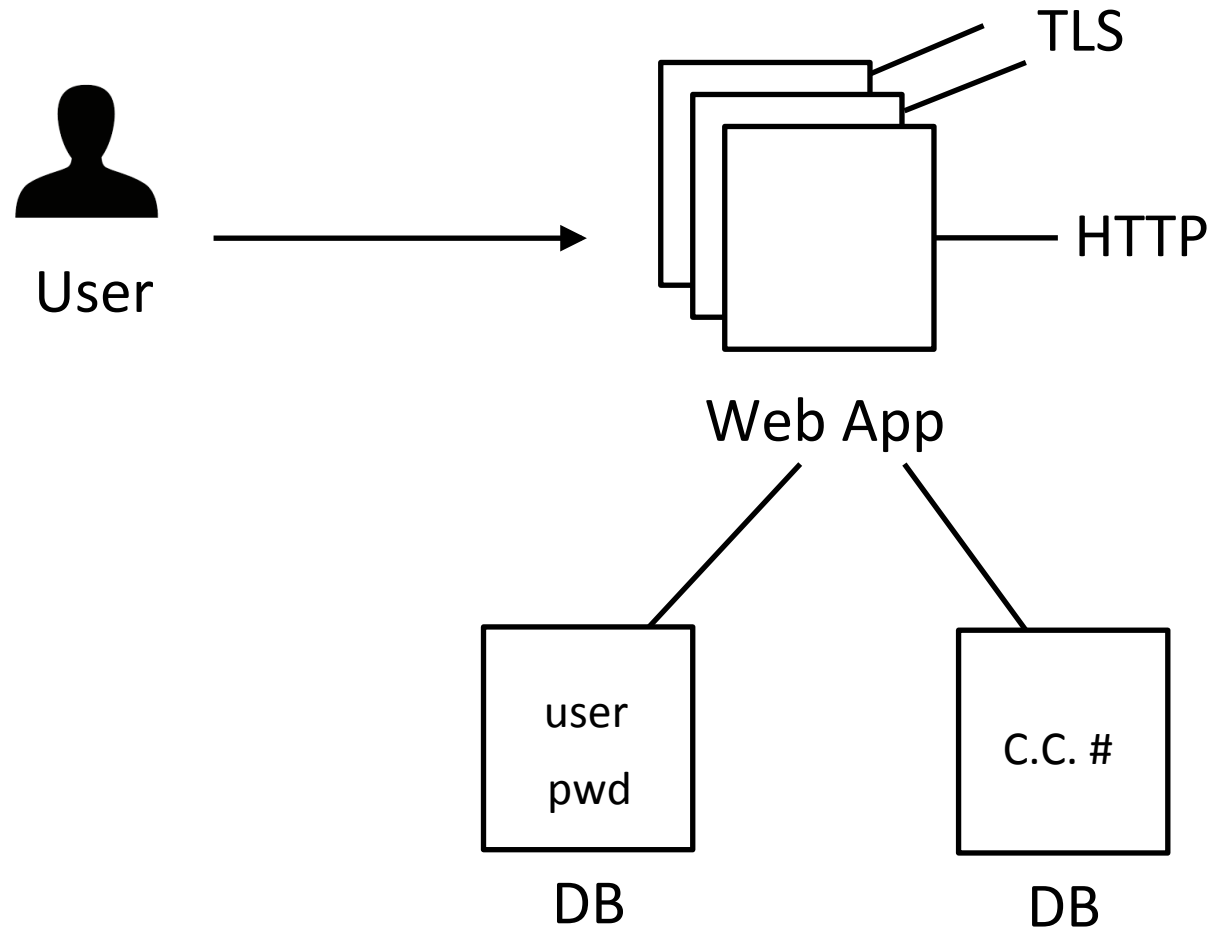
Cloudbleed

```
356 [REDACTED]
357 [REDACTED]
358 GET /1/user/-/activities/activityLevels/date/[REDACTED] HTTP/1.1
359 CF-RAY: [REDACTED]
360 FL-Server: [REDACTED]
361 Host: iphone-cdn-api.fitbit.com
362 X-Real-IP: [REDACTED]
363 Accept-Encoding: gzip
364 Client-Accept-Encoding: gzip, deflate
365 X-Forwarded-Proto: https
366 Connect-Via-Https: on
367 Connect-Via-Port: 443
368 Connect-Via-IP: [REDACTED]
369 Connect-Via-Host: iphone-cdn-api.fitbit.com
370 CF-Visitor: {"scheme":"https"}
371 CF-Host-Origin-IP: [REDACTED]
372 Zone-ID: [REDACTED]
373 Owner-ID: [REDACTED]
374 CF-Int-Brand-ID: [REDACTED]
375 Zone-Name: fitbit.com
376 Connection: Keep-Alive
377 X-SSL-Protocol: TLSv1.2
378 X-SSL-Cipher: ECDHE-RSA-AES128-GCM-SHA256
379 X-SSL-Server-Name: iphone-cdn-api.fitbit.com
380 X-SSL-Session-Reused: .
381 X-SSL-Server-IP: [REDACTED]
382 X-SSL-Connection-ID: [REDACTED]
383 X-SSL-ClientHello: [REDACTED]
384 X-SPDY-Protocol: [REDACTED]
385 accept: */*
386 content-type: application/x-www-form-urlencoded; charset=UTF-8
387 x-app-version: [REDACTED]
388 accept-locale: [REDACTED]
389 authorization: [REDACTED]
390 user-agent: FitbitMobile/2. [REDACTED] (iPhone; iOS 10 [REDACTED]; Scale/[REDACTED])
391 accept-language: [REDACTED]
392 CF-Use-OB: [REDACTED]
393 Set-Expires-TTL: 14400
394 CF-Cache-Max-File-Size: 512m
395 Set-SSL-Name: iphone-cdn-api.fitbit.com
396 CF-Cache-Level: byc
397 CF-Unbuffered-Upload: 0
398 Set-SSL-Client-Cert: 0
399 Set-Limit-Conn-Cache-Host: 50000
400 CF-WAN-RG5: 0
401 CF-Brand-Name: cloudflare
```

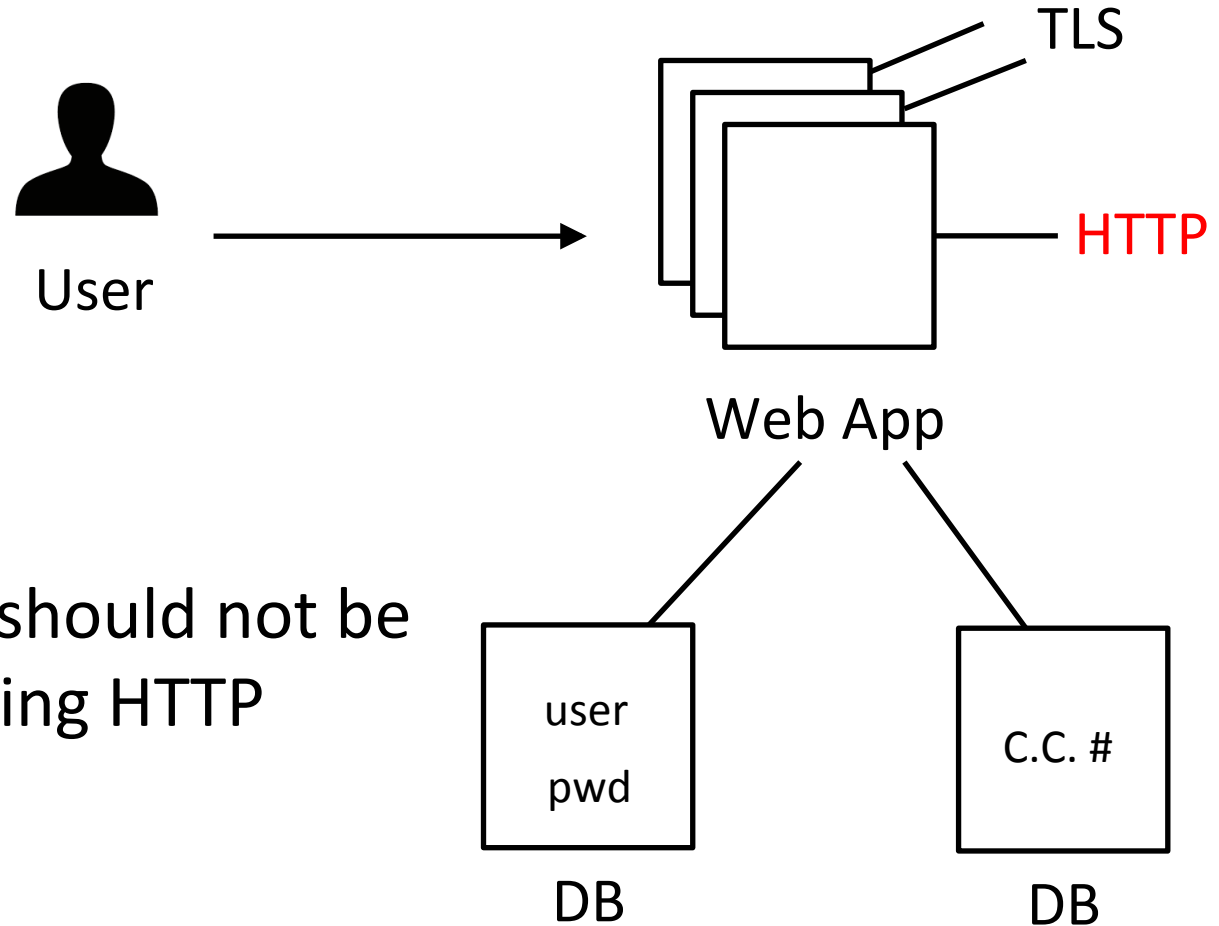
Cloudbleed

- Started on 22 September 2016
- The impact was tremendous because nearly 6 million websites use Cloudflare's services.
- Between 22 September 2016 and 18 February 2017 the bug was triggered 1,242,071 times

Bottomline Scenarios

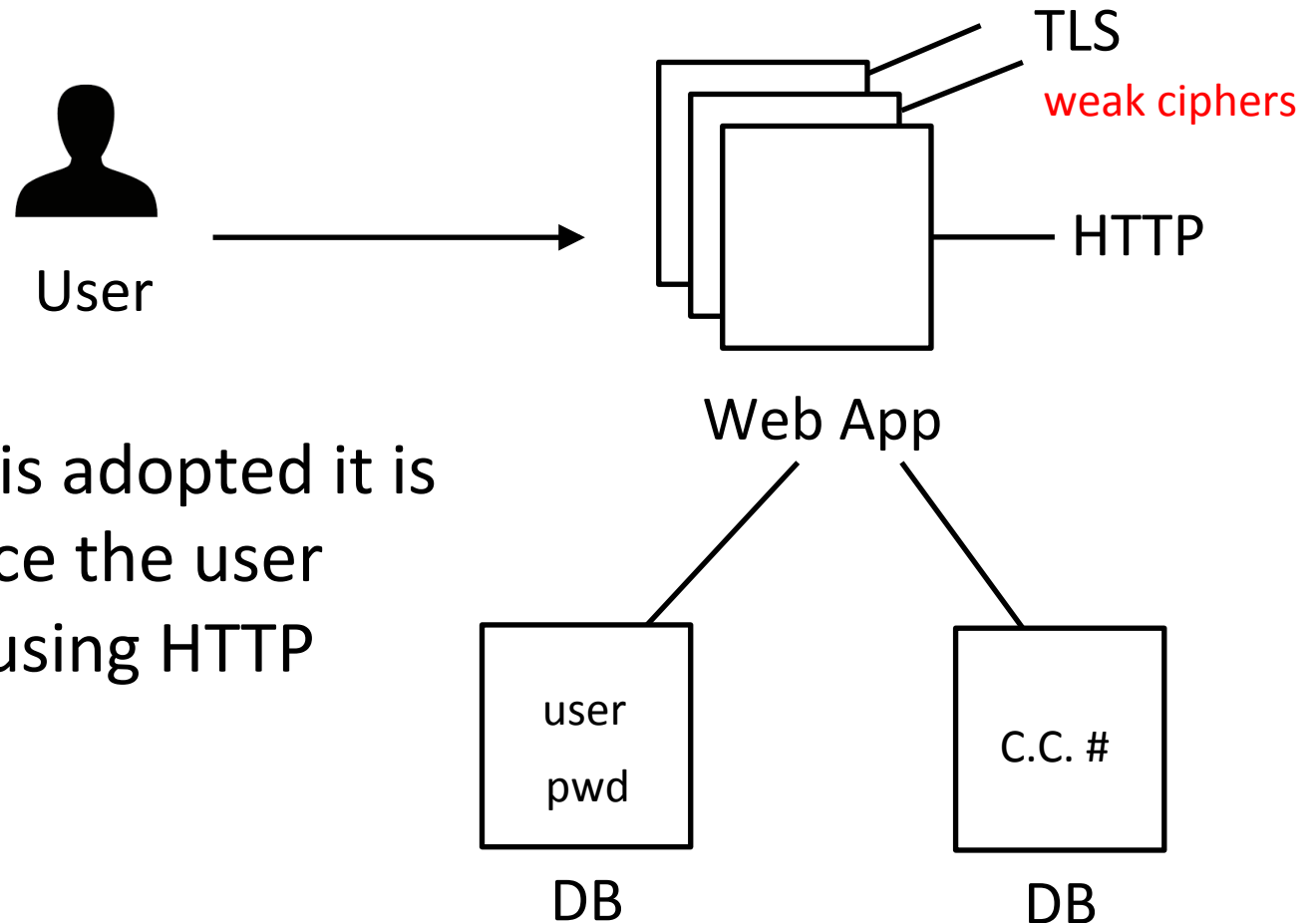


Scenario 1



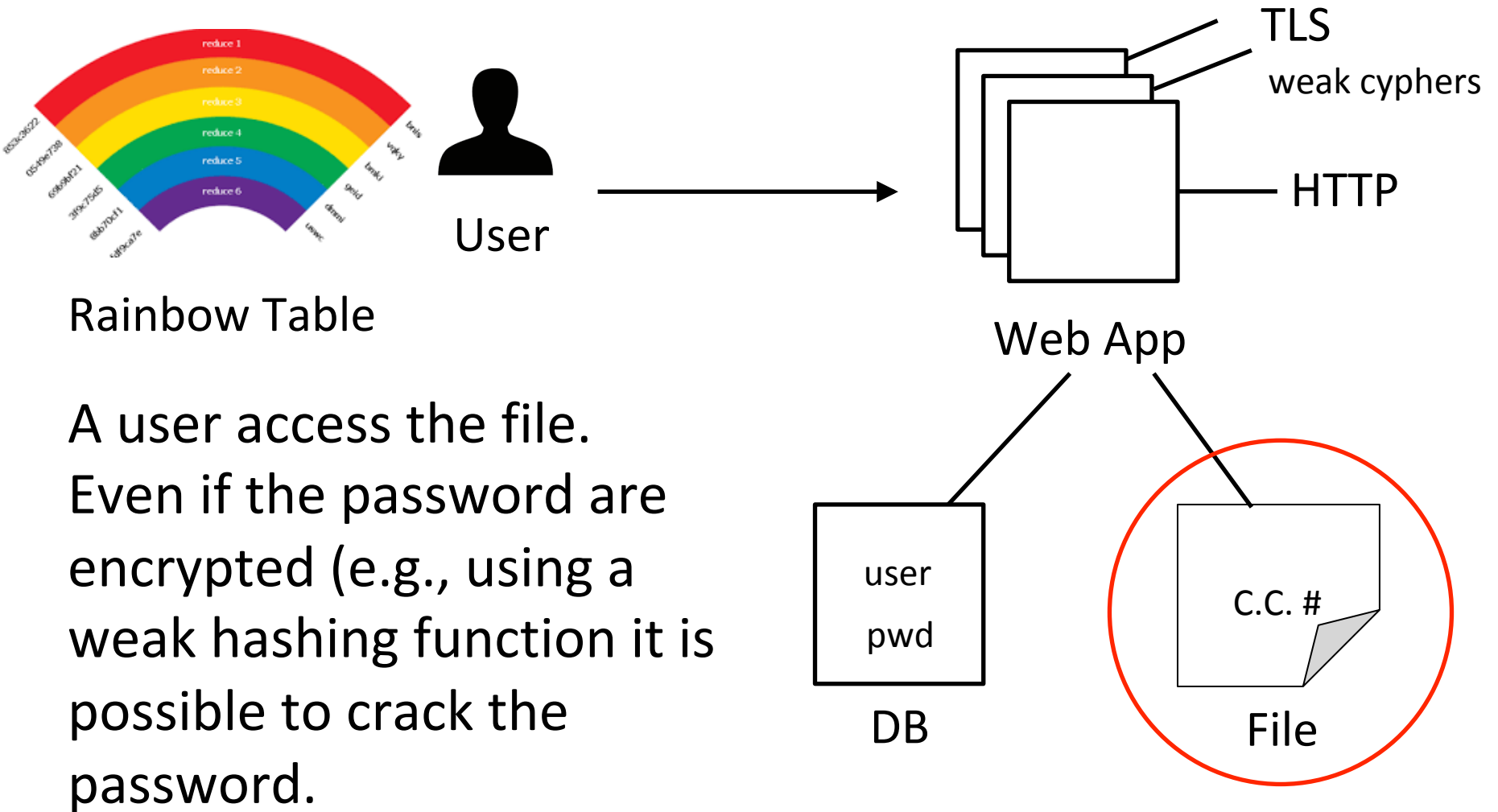
Sensitive data should not be transmitted using HTTP

Scenario 2



If weak cipher is adopted it is possible to force the user transmit data using HTTP

Scenario 3



A user access the file.
Even if the password are encrypted (e.g., using a weak hashing function it is possible to crack the password.

Rainbow Table

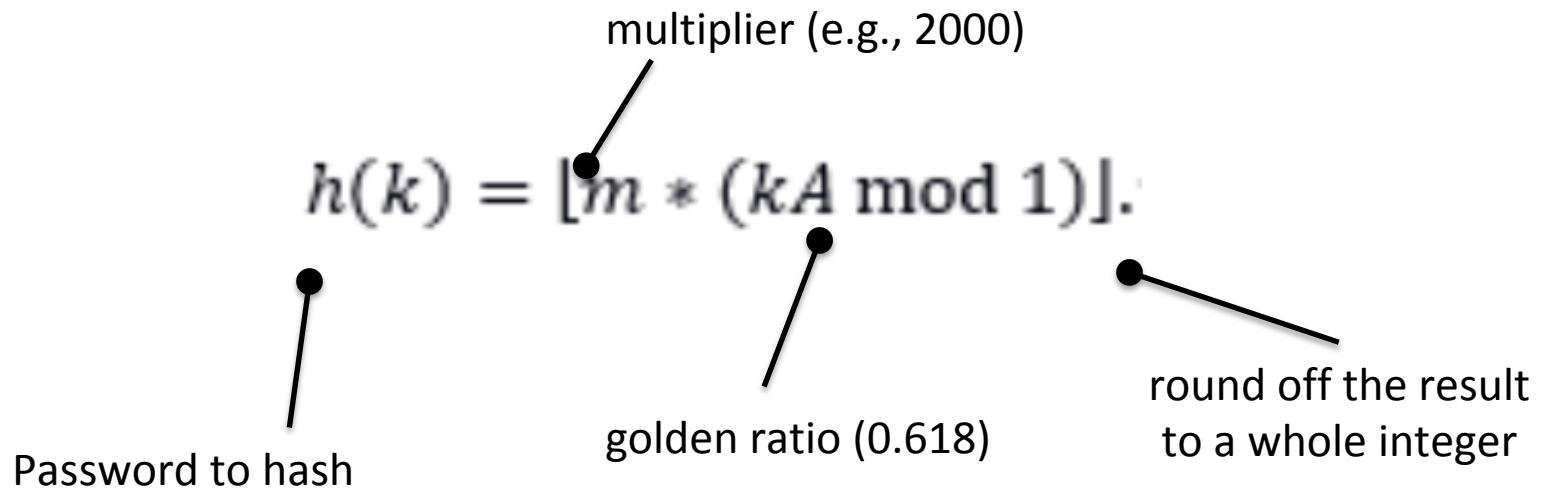
- Huge sets of pre-computed tables filled with hash values that are pre-matched to possible plaintext passwords.
- Based on the tradeoff time VS space

You can create your own rainbow table!

Rainbow Table - Example

Imagine we use the following hash function

Multiplication



The diagram shows the hash function $h(k) = [m * (kA \bmod 1)]$ with three annotations. A line points from the text 'Password to hash' to the variable k in the function. Another line points from the text 'multiplier (e.g., 2000)' to the variable m . A third line points from the text 'golden ratio (0.618)' to the variable A . A fourth line points from the text 'round off the result to a whole integer' to the floor function brackets $[]$.

$$h(k) = [m * (kA \bmod 1)]$$

multiplier (e.g., 2000)

golden ratio (0.618)

round off the result to a whole integer

Password to hash

Rainbow Table - Example

We have a character set with only numbers and 2 places

$$h(k) = \lfloor 2000 * (78 * 0,618 \bmod 1) \rfloor = \lfloor 2000 * (48,204 \bmod 1) \rfloor = \lfloor 2000 * 0,204 \rfloor = \lfloor 408 \rfloor.$$

Password	Hash value
Null	Null
01	1236
02	0472
03	1708
...	...
78	0408
...	...
99	0364

Reducing the table

Only the last 2 digits of the password are kept

p1	h1	p2	h2	p3	h3	p4	h4
Null	Null	Null	Null	Null	Null	Null	Null
01	1236	36	0496	96	0656	56	1215
02	0472	72	0992	92	1712	12	0832
03	1708	08	1888	88	0768	68	0048
04	0944	44	0384	84	1824	24	1664
05	0180	80	0879	79	1644	44	0384

Reducing the table

Only the last 2 digits of the password are kept

p1	h1	p2	h2	p3	h3	p4	h4
Null	Null	Null	Null	Null	Null	Null	Null
01	1236	36	0496	96	0656	56	1215
02	0472	72	0992	92	1712	12	0832
03	1708	08	1888	88	0768	68	0048
04	0944	44	0384	84	1824	24	1664
05	0180	80	0879	79	1644	44	0384

Only the left- and the right-most columns are kept

Reducing the table

p1	h4
Null	Null
01	1215
02	0832
03	0048
04	1664
05	0384
06	1260
07	0656
09	0944
10	0607

Using the Rainbow Table

We are looking for hash value 1888

- 1888 Not in the hashes
- 1888 → (r) 88
- 88 not in the passwords
- 88 → (h) 0768
- 0768 not in the hashes
- 0768 → (r) 68
- 68 not in the passwords
- 68 → (h) 0048

p1	h4
Null	Null
01	1215
02	0832
03	0048
04	1664
05	0384
06	1260
07	0656
09	0944
10	0607

Using the Rainbow Table

We are looking for hash value 1888

- 1888 Not in the hashes
- 1888 \rightarrow (r) 88
- 88 not in the passwords
- 88 \rightarrow (h) 0768
- 0768 not in the hashes
- 0768 \rightarrow (r) 68
- 68 not in the passwords
- 68 \rightarrow (h) 0048

p1	h4
Null	Null
01	1215
02	0832
03	0048
04	1664
05	0384
06	1260
07	0656
09	0944
10	0607

- From 03 \rightarrow (h) 1708

Using the Rainbow Table

We are looking for hash value 1888

- 1888 Not in the hashes
- 1888 \rightarrow (r) 88
- 88 not in the passwords
- 88 \rightarrow (h) 0768
- 0768 not in the hashes
- 0768 \rightarrow (r) 68
- 68 not in the passwords
- 68 \rightarrow (h) 0048

p1	h4
Null	Null
01	1215
02	0832
03	0048
04	1664
05	0384
06	1260
07	0656
09	0944
10	0607

- From 03 \rightarrow (h) 1708 \rightarrow (r) 08 \rightarrow (h) 1888 Original value we were looking for

Sensitive Data Exposure

A3:2017 – Sensitive Data Exposure

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	⊗	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	⊗	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

A3:2017 – Sensitive Data Exposure

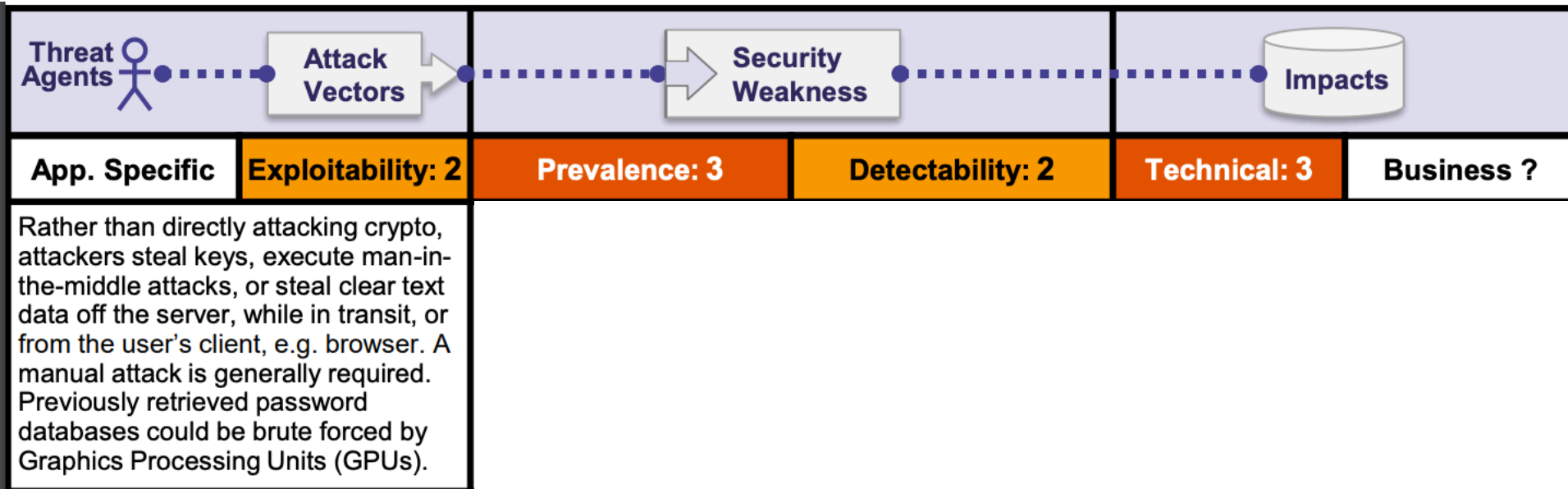
A3:2017- Sensitive Data Exposure

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

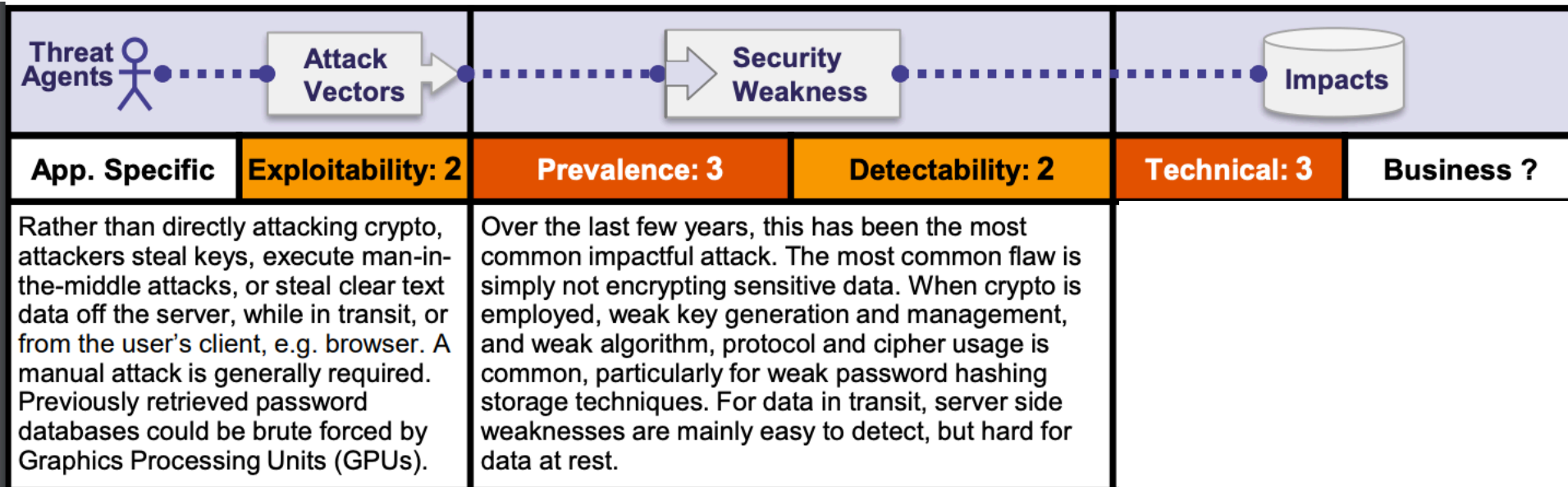
A3:2017 – Sensitive Data Exposure



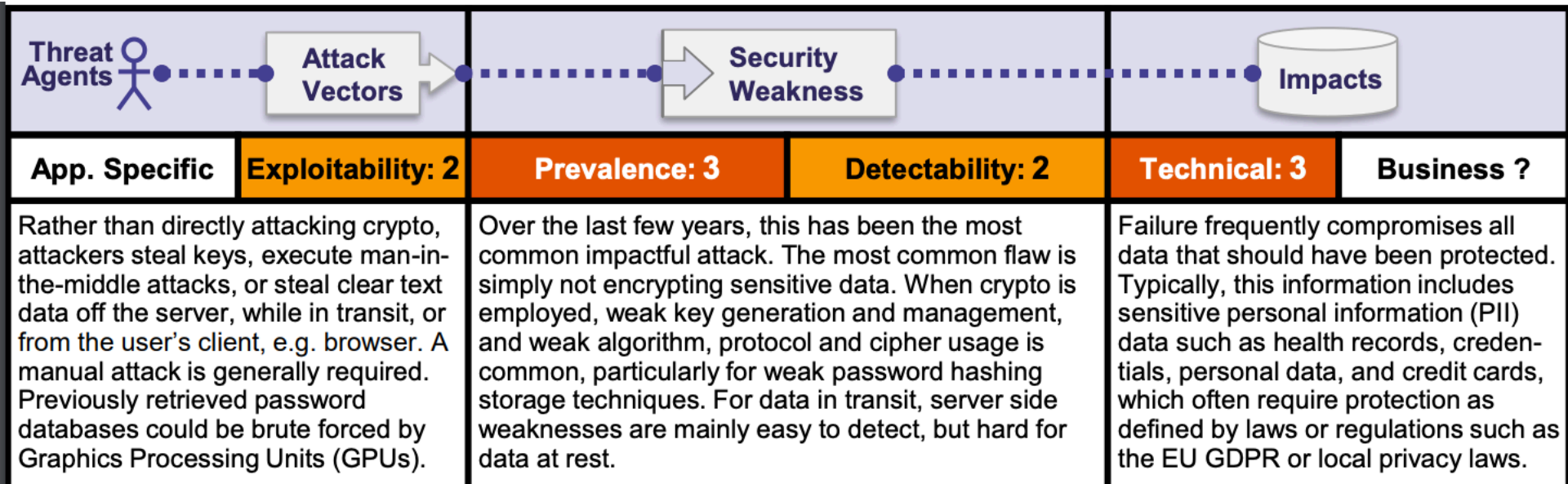
A3:2017 – Sensitive Data Exposure



A3:2017 – Sensitive Data Exposure



A3:2017 – Sensitive Data Exposure



Testing for Sensitive Data Exposure

Is The Application Vulnerable? (1/3)

Determine protection needs of data in transit and at rest, particularly if data falls under privacy laws (e.g., GDPR).

- Example of what may be a sensitive data: passwords, credit card numbers, health records, personal information, business secrets, ...

Is The Application Vulnerable? (2/3)

Is any data *transmitted* in clear text?

- This especially concerns protocols such as HTTP, SMTP, and FTP
- External internet traffic is dangerous, e.g. between load balancers, web servers, or back-end systems

Is sensitive data stored in clear text including backups?

Are any old or weak cryptographic algorithms used either by default or in older code?

Is The Application Vulnerable? (3/3)

Are default crypto keys in use, weak crypt keys generated or re-used, or is proper key management or rotation missing?

Is encryption not enforced, e.g. are any user agent (browser) security directives or header missing?

Does the user agent (e.g., app, mail client) not verify if the received server certificate is valid?

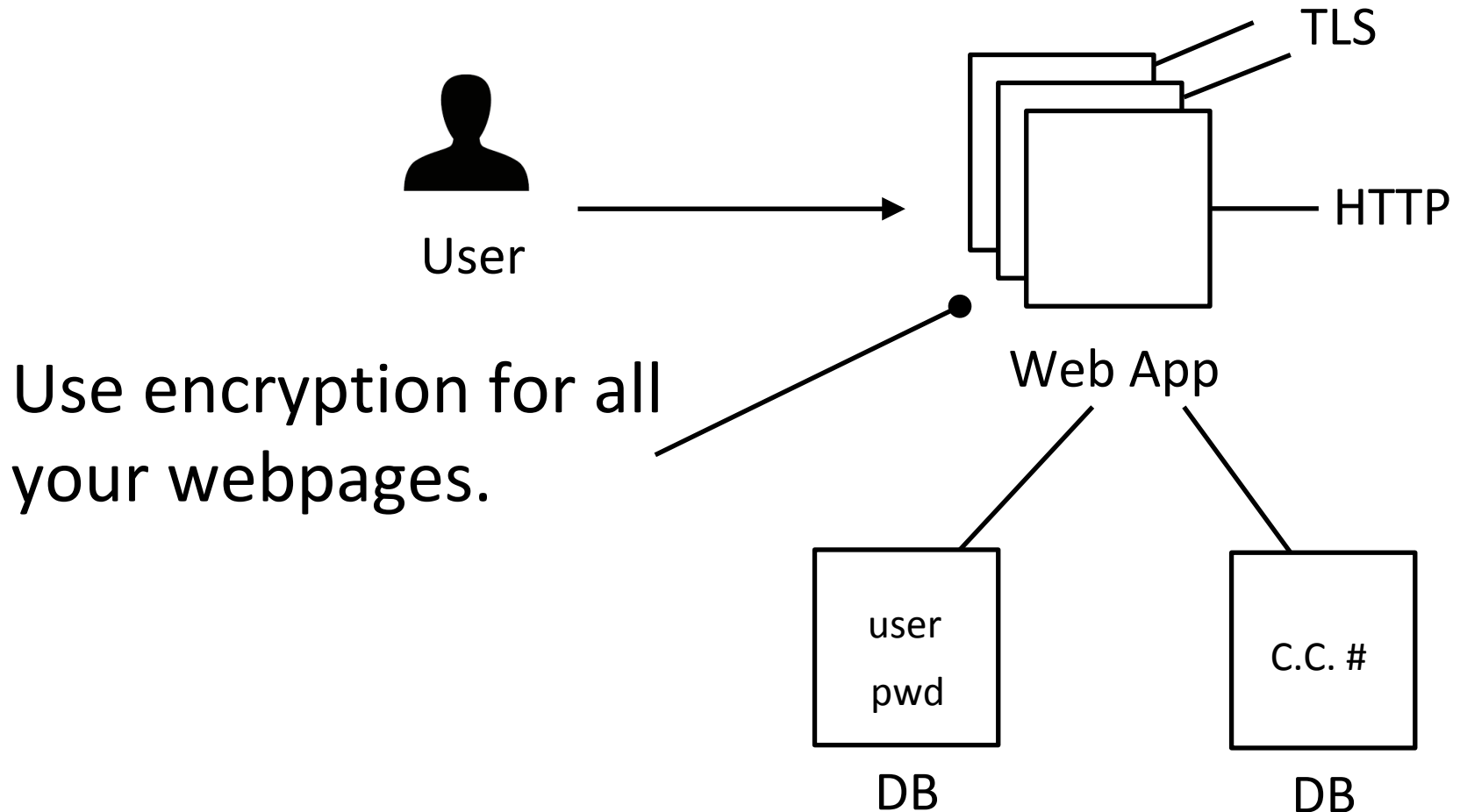
Insecure Communication Exercise (WebGoat)

Instructions

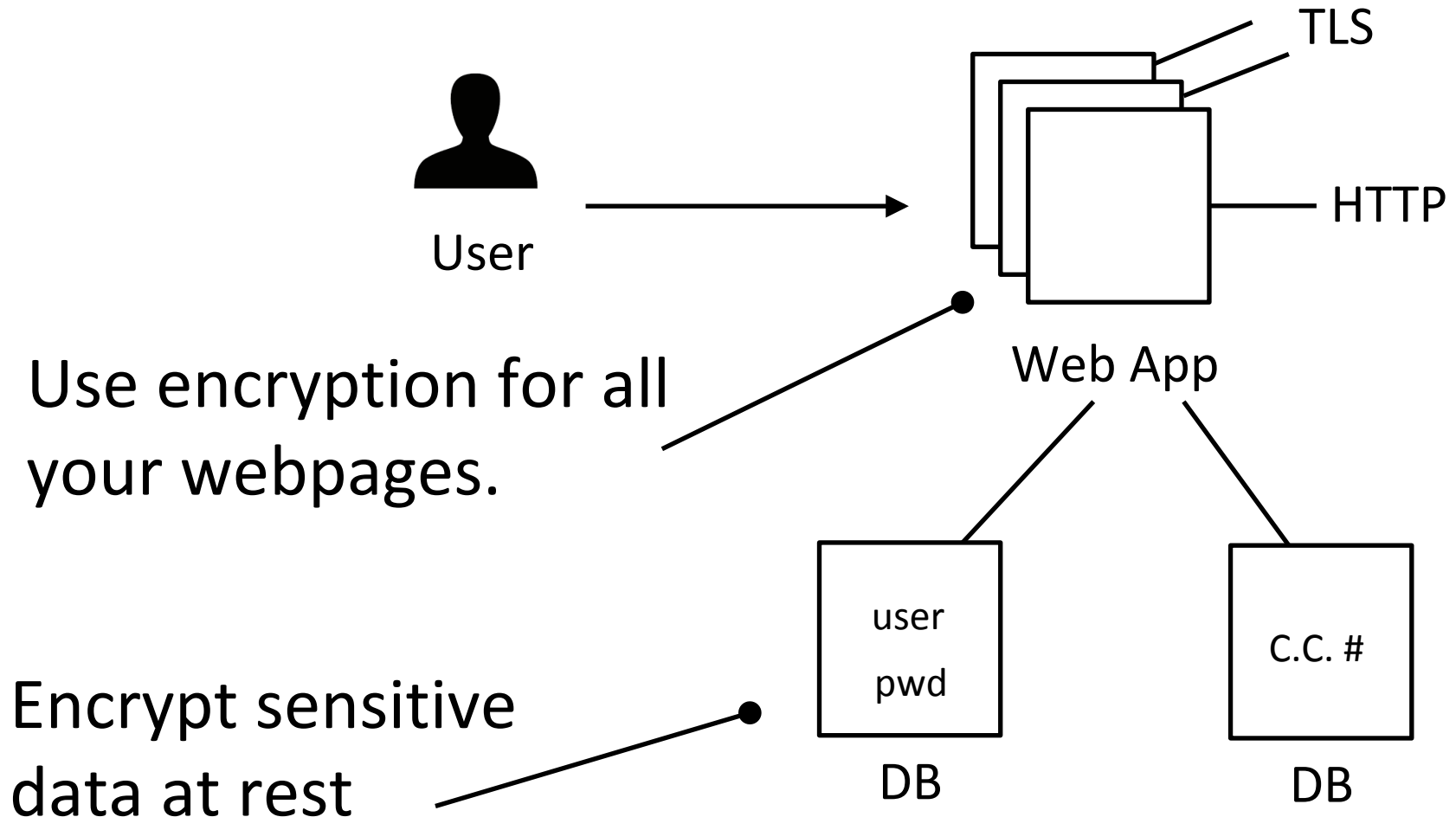
- Open WebGoat and select “Insecure Login”
- To sniff the network traffic you can use Wireshark
- You should be able to see the username and password in clear-text that are necessary to perform the login
- An example of packet capture is available on Moodle

Countermeasures

Countermeasures



Countermeasures



Countermeasures

- Classify data processed, stored, or transmitted by an application.
- Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
- Apply controls as per the classification.

Examples of Sensitive Data (1/2)

- Customer information
 - Names, home addresses, payment card information, social security numbers, emails,...
- Employee data
 - Banking information, username/password
- Intellectual properties and trades secrets
- Business operations and/or inventory

Examples of Sensitive Data (2/2)

- The GDPR considers the following data as personal data (cannot be processed without explicit consent):
 - Race and ethnicity
 - Political, religious or philosophical beliefs including union membership
 - Genetic and biometric data (for the purpose of uniquely identification).

Countermeasures

- Encrypt data at rest
- Use strong ciphers at your web server
 - Key exchange: Diffie-Hellman key exchange with minimum 2048 bits
 - Message Integrity: HMAC-SHA2
 - Message Hash: SHA2 256 bits
 - Asymmetric encryption: RSA 2048 bits
 - Symmetric encryption: AES 128 bits
 - Password Hashing: Argon2, PBKDF2, Scrypt, Bcrypt

Strong salted hash functions



Salting & Hashing of Passwords

- Salting is a technique in which we add a random string to the user entered password and then hash the resulting string.
- Even if two people have chose the same password, the salt for them will be different

User	Password	Random Salt	Resultant Password String	Hash
user1	one	12	one12	1234
user2	two	23	two23	2345
user3	three	34	three34	3456
user4	one	45	one45	4567
user5	two	56	two56	5678

Password Creation Process

- User enters a password
- A random salt value is generated of the user
- The salt value is added to the password and a final string is generated
- The hash for the final string is calculated
- The hash and the salt are stored in the database for the user

User tries to login

- User enters his user id
- The user is used to retrieve the password hash and salt stored in the database
- The user enters his password
- The retrieved salt is added to this password and a final string is generated
- The hash for the final string is calculated
- If it matches the password it is correct, otherwise it is not.

Importance of Salting

				
Password	p4s5w3rdz	p4s5w3rdz	p4s5w3rdz	p4s5w3rdz
Salt	-	-	et52ed	ye5sf8
Hash	f4c31aa	f4c31aa	1vn49sa	z32i6t0

- Use cryptographically strong random data to generate the salt
- Generate a unique salt for each sensitive data

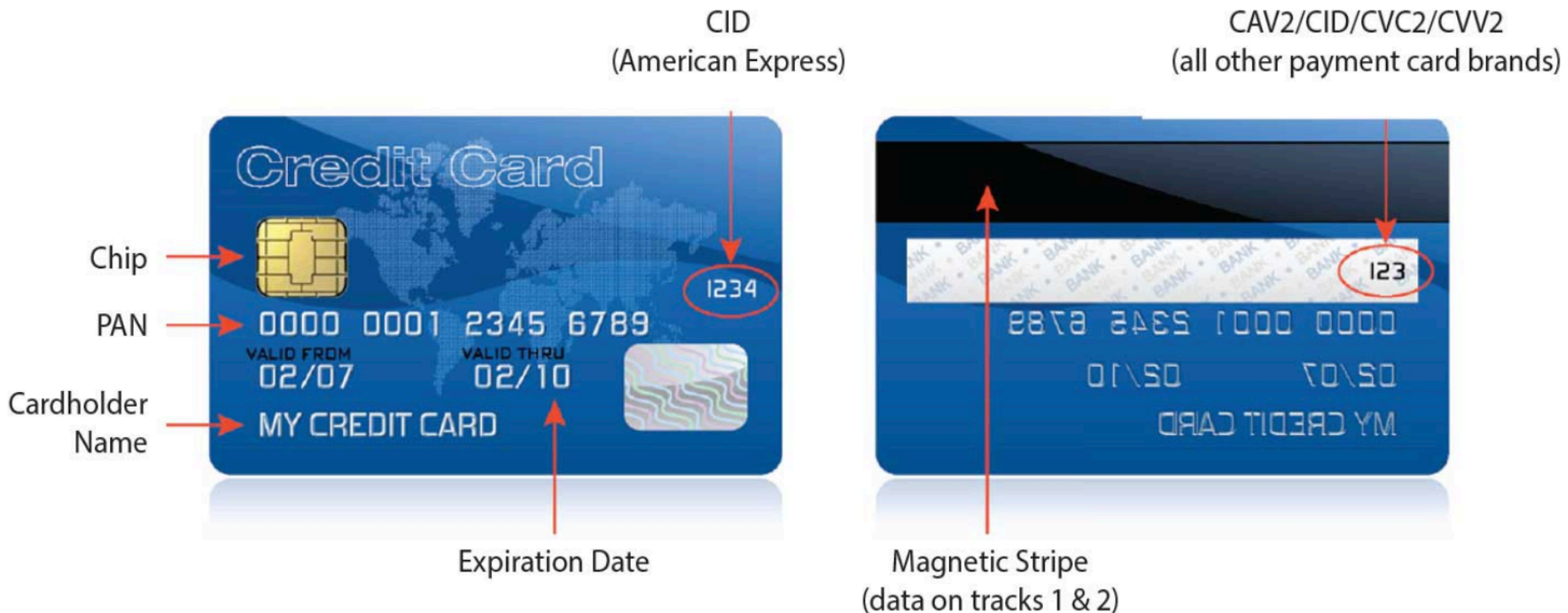
Countermeasures

- Don't store sensitive data except if you have to.
- Discard it as soon as possible or use PCI DSS (Payment Card Industry Data Security Standard) compliant tokenization or even truncation.

PCI –DSS Truncation

- Limit visibility of the PAN (Primary Account Number) to the first 6 and the last 4 digits

Types of Data on a Payment Card



PCI-DSS Tokenization

Tokenization as used within this document is a process by which a surrogate value, called a “token,” replaces the primary account number (PAN) and, optionally, other data. The tokenization process may or may not include functionality to exchange a token for the original PAN (“de-tokenization”). The security of an individual token relies predominantly on the infeasibility of determining the original PAN knowing only the surrogate value (i.e., token).

PAN	Token	Comment
3124 005917 23387	7aF1Zx118523mw4cwl5x2	Token consists of alphabetic and numeric characters
4959 0059 0172 3389	729129118523184663129	Token consists of numeric characters only
5994 0059 0172 3383	599400x18523mw4cw3383	Token consists of truncated PAN (first 6, last 4 of PAN are retained) with alphabetic and numeric characters replacing middle digits.