Probabilistic attacks against compressed encrypted protocols

Dimitrios Karakostas **Dionysis Zindros** **Aristeidis Pagourtzis**

Theoretical background
(gzip)

- **gzip**: The most used encryption software in the Internet.

- Implements the DEFLATE algorithm:

$$DEFLATE(m) = Huffman(LZ77(m))$$

Theoretical background
(LZ77)

- **LZ77**: Lossless data compression algorithm, published in 1977 by A. Lempel and J. Ziv.
- Method:
  - Find repeated portions of data.
  - Replace them with references as [length, offset].
  - Minimum length = 3.
  - Maximum offset = 32Kb.

Theoretical background
(LZ77)

LZ77 example

Hello, world! I love you.

Hello, world! I hate you.

Hello, world! Hello world! Hello world!

Hello, world! I love you.

(**26**, **16**) **hate** (**21**, **5**)

(**26**, **14**) (**14**, **28**)

Theoretical background
(Huffman)

- **Huffman coding**: Lossless data compression algorithm, proposed by D. Huffman in 1952.
- Method:
  - Analyze the frequency of each letter in the text.
  - Replace common letters with short codes.
  - Replace rare letters with long codes.
  - Code alphabet should be prefix free.

Theoretical background
(Huffman)

Huffman example

- Frequency analysis:

o: 6	**n**: 5	**r**: 3	**l**: 3
b: 3	**c**: 3	**a**: 3	**s**: 2
k: 2	**e**: 2	**i**: 2	**f**: 2
h: 1	**d**: 1	**t**: 1	**u**: 1

- Code alphabet:

Short

o: 00	**n**: 01	**r**: 1000	**l**: 1001
b: 1010	**c**: 1011	**a**: 11000	**s**: 11001
k: 11010	**e**: 11011	**i**: 11100	**f**: 1111000
h: 1111001	**d**: 1111010	**t**: 1111011	**u**: 1111100

Long

Theoretical background
(Same-origin policy)

- **Same-origin policy**: scripts in one page are allowed to access data in a second page if both have the same origin.
- Origin: protocol, host and port of a URL.
- Documents retrieved from distinct origins are isolated from each other.
- i.e. a document retrieved from http://example.com/target.html is disallowed to access the DOM of a document retrieved from https://head.example.com/target.html.

Theoretical background
(Same-origin policy)

- Attacks on same-origin policy:
  - Cross-site scripting (XSS): vulnerability that allows an attacker to inject a client-side script into web pages viewed by other users.
  - Cross-site request forgery (CSRF): exploit that allows the attacker to issue unauthorized requests to a website, on behalf of a user the website trusts.

Theoretical background
(TLS)

- **Transport Layer Security (TLS)**: protocol that provides security over the internet.
- Prevents eavesdropping, tampering or message forgery.
- TLS handshake allows the negotiation of a symmetric key via asymmetric cryptography, provided by certificates created by trusted authorities.

Theoretical background
(TLS)

- TLS record structure

+	Byte +0	Byte +1	Byte +2	Byte +3
Byte 0	Content type			
Bytes 1..4	Version		Length	
	(Major)	*(Minor)*	*(bits 15..8)*	*(bits 7..0)*
Bytes 5..(*m*-1)	Protocol message(s)			
Bytes *m*..(p-1)	MAC (optional)			
Bytes *p*..(q-1)	Padding (block ciphers only)			

Theoretical background
(MitM)

- Man-in-the-Middle: one of the most common attack vectors on modern communications.

- Common MitM techniques:
  - ARP Spoofing: the attacker sends ARP messages, so that its MAC address is associated with the target endpoint's IP address.
  - DNS Poisoning: the attacker introduces data into a DNS resolver's cache, to return incorrect address for the chosen endpoint.

IND-PCPA
(PCPA game)

- Traditionally, cryptographers have used games for security analysis
- IND-CPA, IND-CCA{1,2}
- We introduce a new security game:

Indistinguishability under partially chosen plaintext attack (IND-PCPA)

IND-PCPA
(PCPA game)

- The challenger generates a pair P_k, S_k and publishes P_k to the adversary.
- The adversary may perform a polynomially bounded number of encryptions or other operations.
- Eventually, the adversary submits two distinct chosen plaintexts M_0, M_1 to the challenger.
- The challenger selects a bit $b \in \{0,1\}$ uniformly at random.
- The adversary can then submit any number of selected plaintexts R_i , $i \in N$, $|R| \geq 0$, and the challenger sends the ciphertext $C_i = E(P_k, M_b || R_i)$ back to the adversary.
- The adversary is free to perform any number of additional computations or encryptions, before finally guessing the value of b.

IND-PCPA
(PCPA game)

A cryptosystem is indistinguishable under partially chosen plaintext attack, if every probabilistic polynomial time adversary has only a negligible advantage on finding b over random guessing.

IND-PCPA

- IND-PCPA vs IND-CPA:
  - The adversary submits the empty string as chosen plaintext.
  - The challenger then sends back:

$$C = E(P_k, M_b || \text{""}) = E(P_k, M_b)$$

  - which is the challenger response of the IND-CPA game.
  - Intuitively, if the adversary can beat the game of IND-PCPA, he also has the ability to beat IND-CPA.

IND-PCPA

- PCPA scenario on compression-before-encryption protocol:
  - A system creates:

    c = Encrypt(Compress(m))

    where c is the ciphertext of the compressed m.
  - The attacker issues a PCPA creating:

    $m = n_1||secret||n_2||reflection||n_3$
  - where n_1,n_2,n_3 are random nonces.
  - If the chosen reflection is the same as the secret, a pattern emerges and the compression is better, possibly resulting in smaller ciphertext, compared to the one of a wrong reflection.

IND-PCPA
(PCPA exploits)

- CRIME:
  - [Rizzo, Duong ‘12]
  - CRIME attacked TLS header compression in HTTPS.
  - TLS header compression is now disabled.
  - CRIME is no longer possible.
  - CRIME set the foundation for compression/encryption attacks.

IND-PCPA
(PCPA exploits)

- BREACH:
  - [Prado, Harris, Gluck ‘13]
  - BREACH was based on CRIME.
  - BREACH attacks HTTPS response.
  - Original BREACH attack had specific assumptions:
    - Against stream ciphers.
    - No noise in response.
    - Secret has known prefix, bootstrapping is trivial.

Attack model
(Assumptions)

- The attacker has gained control of the victim's network and can view the victim's encrypted traffic, which can be accomplished by MitM.
- The attack script issues requests toward the chosen endpoint from the victim's browser, i.e. via XSS.
- Each request contains a chosen stream of data, which is reflected in the response body, along with the secret.
- Compression is applied on both the secret and the reflection.

Attack model

- MitM implementation:
  - We add a rule in the hosts file of the lab machine, in order for all traffic toward an endpoint to be redirected to the localhost.
  - We implemented a Python MitM proxy, that opens TCP sockets on both the lab machine and the endpoint and forwards traffic on both ends, while parsing the header and (encrypted) body TLS record.
  - We also implemented a defragmentation mechanism, in order to parse records that span over multiple TCP packets.

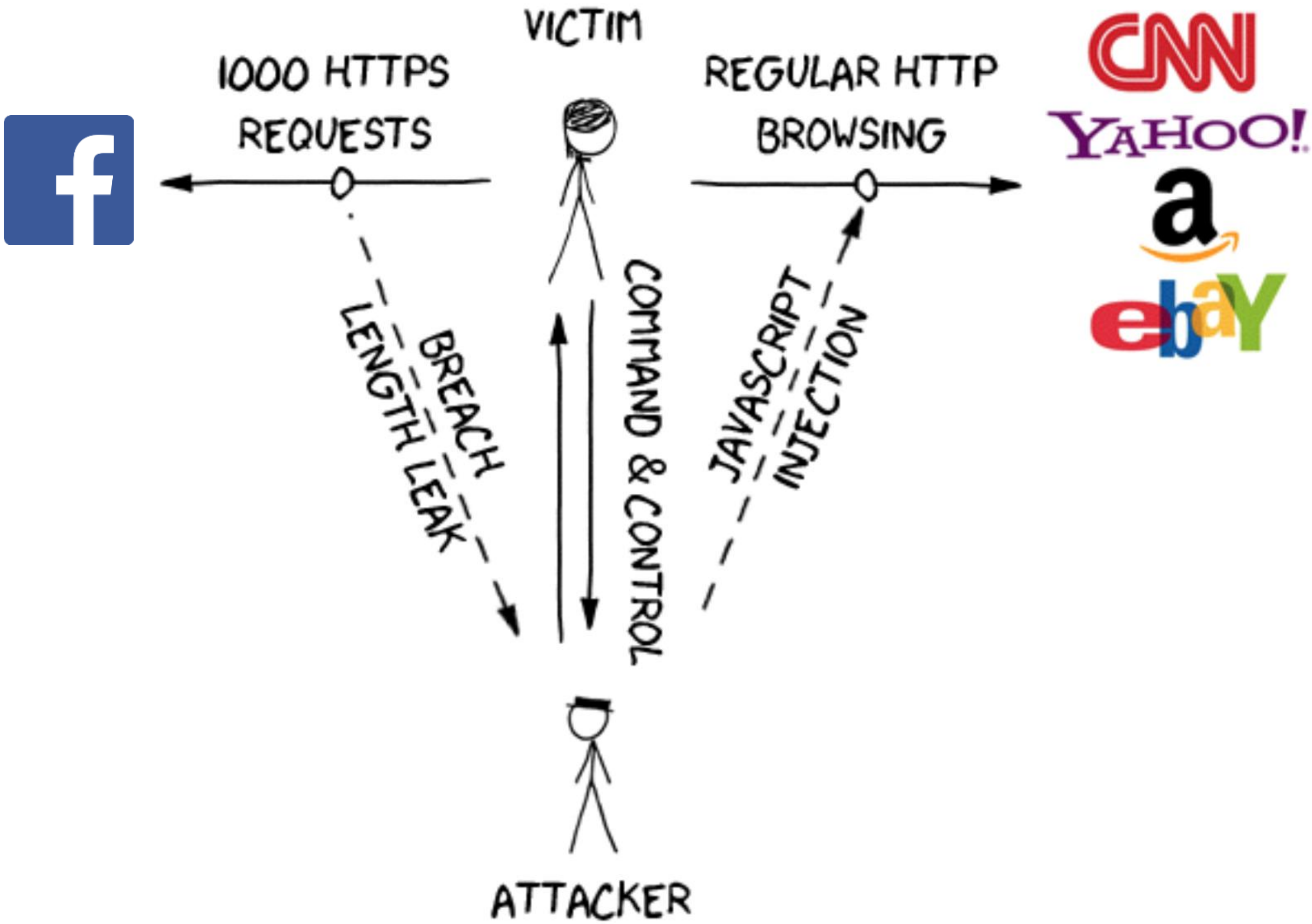
Attack model

- BREACH script implementation:
  - The user inputs a known prefix for the secret, needed to bootstrap the attack, and the alphabet that the characters of the secret belong.
  - An attack vector is created, with each item corresponding to a fragment of the alphabet, where the sum of the fragment makes up the whole alphabet.
  - A request is issued for each item of the vector every 4 seconds, resuming from the beginning when the end of the vector is reached.
  - The requests are made in the form of <img> tags, injected in the HTML body of a controlled website.

Attack model

- Attack persistence:
  - We propose a command-and-control mechanism that allows the execution of the attack without the need of a contaminated website, that the victim would visit.
  - The victim needs to browse the HTTP web.
  - The attacker that controls the victim's traffic would inject the attack script in the response from a regular HTTP website.

Attack model



Attack model

- Vulnerable endpoints:
  - Facebook Chat messages
  - Gmail Authentication token
  - Gmail private emails

Attack model
(Facebook Chat messages)

- Facebook Chat messages:
  - Facebook provides a lightweight mobile version, Facebook Touch.
  - It also allows a search functionality via URL, in the form:

    https://touch.facebook.com/messages?q=<search_string>
  - The search string is reflected in the body of the response.
  - Also, regardless of the search results, the last message of the 5 most recent conversations is also included in the body.

Attack model

(Facebook Chat messages)

Attack model
(Gmail Authentication token)

- Gmail Authentication token:
  - Gmail provides a plain HTML version for faster browsing, which enables a search functionality as:

    https://mail.google.com/mail/u/0/x/?s=q&q=<search_string>
  - Each request should contain a valid, random-generated string between the *0* and *x* parameter of the URL.
  - If no string is included, a redirection to a URL that contains such a string is applied, returning an empty result page, stating the action as incomplete.
  - However, the HTML body contains both the search string and the authentication token for the account.
  - Different tokens of different accounts demonstrate a fixed prefix: “AF6bup”.

Attack model

(Gmail Authentication token)

```
amp;pv=tl&amp;eot=1&
amp;q=ladbfsk!1_3_5_7_9_b_d_f_h_j_l_n_p_r_t_v_x_z_B_D_F_H_J_L_N_P_R_T_V_X_Z___2_4
_6_8_a_c_e_g_i_k_m_o_q_s_u_w_y_A_C_E_G_I_K_M_O_Q_S_U_W_Y_-_AF6bup0znq&amp;v=b&
amp;s=q">Compose</a></td></tr></table><div class="notification">We cannot
complete the action at this time. Please try again using the search action above.
</div><form action="?&amp;mnut=tl&amp;v=mnu" name="f" method="post"><input
type="hidden" name="at" value="AF6bupNx9G8BD_Wr7frvMfpnjj_Nh_0GVQ" /><input
type="hidden" name="nact" value="?&amp;at=AF6bupNx9G8BD_Wr7frvMfpnjj_Nh_0GVQ&
amp;s=q" /><input type="hidden" name="nredir" value="?&
amp;q=ladbfsk!1_3_5_7_9_b_d_f_h_j_l_n_p_r_t_v_x_z_B_D_F_H_J_L_N_P_R_T_V_X_Z___2_4
_6_8_a_c_e_g_i_k_m_o_q_s_u_w_y_A_C_E_G_I_K_M_O_Q_S_U_W_Y_-_AF6bup0znq&amp;s=q"
/><input type="hidden" name="search" value="query" /><div class="noMatches">No
results for:
ladbfsk!1_3_5_7_9_b_d_f_h_j_l_n_p_r_t_v_x_z_B_D_F_H_J_L_N_P_R_T_V_X_Z___2_4_6_8_a
_c_e_g_i_k_m_o_q_s_u_w_y_A_C_E_G_I_K_M_O_Q_S_U_W_Y_-_AF6bup0znq</div><script
type="text/javascript">
var token="AF6bupNx9G8BD_Wr7frvMfpnjj_Nh_0GVQ";var
searchPageLinks=document.getElementsByClassName("searchPageLink");
for(i=0;i<searchPageLinks.length;i++)searchPageLinks[i].onclick=function(e){var
href=e.currentTarget.href;var form=document.createElement("form");
form.setAttribute("method","post");form.setAttribute("action",href);var
inputToken=document.createElement("input");
```

Reflection

Authentication token

Attack model
(Gmail private emails)

- Gmail private emails:
  – The attacker issues a search request through a URL like:

    https://mail.google.com/mail/u/0#search/<search_string>

  – The response body does not include the search string, however, it contains both the Subject and a fragment of the body of the latest inbox mails.
  – The attacker could send multiple mails to the victim, that would be included in the response, along with other private mails.

Attack model

- Validation of secret-reflection compression:
  - We use mitmproxy[1], to extract the compressed body of a response that was obtained with the attack.
  - We use infgen[2], to disassemble the compressed body to the LZ77 compression of the initial data stream.

```
match 21 378
match 8 325
literal 'potrymmkwi_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_
match 4 19121
literal 'j
match 3 79
literal 'p
match 4 24700
match 6 16724
match 3 85
match 6 3090
match 8 1330
```

Bad compression

```
match 21 378
match 8 325
literal 'potrymmkwi_a_b_c_d_e_f_g_h_i_j_k_l_m_n_o_p_q_r_s_t_u_v_w_x_y_z_
match 5 19124
match 3 79
literal 'p
match 4 24724
match 6 16727
match 3 85
match 6 3090
match 8 1330
literal 's...
```

Good compression

1. https://mitmproxy.org
2. http://www.zlib.net/infgen.c.gz

Statistical methods
(Block ciphers)

- Original attacks assumed stream ciphers. e.g. original BREACH assumed RC4.
- [Prado, Neal, Gluck] suggested block ciphers are vulnerable, but did not provide practical attack details.
- In this work, we perform practical attacks against popular block ciphers:
  - We attack AES_128 used in Facebook, Gmail, Twitter, Wikipedia, YouTube, Amazon etc.
- We have found that the AES implementation in the NSS library displays certain patterns.

Statistical methods
(Block ciphers)

```
User application payload: 1083
Endpoint application payload: 40
Endpoint application payload: 1524
Endpoint application payload: 101
Endpoint application payload: 1524
Endpoint application payload: 1104
Endpoint application payload: 1524
Endpoint application payload: 2604
Endpoint application payload: 1351
User application payload: 40
User application payload: 1083
Endpoint application payload: 40
Endpoint application payload: 1524
Endpoint application payload: 101
Endpoint application payload: 1524
Endpoint application payload: 1104
Endpoint application payload: 1524
Endpoint application payload: 2604
Endpoint application payload: 1353
User application payload: 40
```

First request

Second request

Facebook flow

```
User application payload: 255
Endpoint application payload: 270
Endpoint application payload: 350
Endpoint application payload: 41
User application payload: 41
User application payload: 259
Endpoint application payload: 74
Endpoint application payload: 1395
Endpoint application payload: 1287
Endpoint application payload: 41
User application payload: 41
User application payload: 255
Endpoint application payload: 271
Endpoint application payload: 402
Endpoint application payload: 41
User application payload: 41
User application payload: 260
Endpoint application payload: 70
Endpoint application payload: 1395
Endpoint application payload: 1304
Endpoint application payload: 41
User application payload: 41
```

First request

First redirection

Second request

Second redirection

Gmail flow

Statistical methods
(Block ciphers)

```
User application payload: 3142
Endpoint application payload: 214
Endpoint application payload: 340
Endpoint application payload: 36
User application payload: 3161
User application payload: 36
Endpoint application payload: 78
Endpoint application payload: 229
Endpoint application payload: 36
User application payload: 36
User application payload: 3015
Endpoint application payload: 53
Endpoint application payload: 1122
Endpoint application payload: 36
User application payload: 36

User application payload: 3142
Endpoint application payload: 80
Endpoint application payload: 340
Endpoint application payload: 36
User application payload: 36
User application payload: 3160
Endpoint application payload: 67
Endpoint application payload: 230
Endpoint application payload: 36
User application payload: 36
User application payload: 3015
Endpoint application payload: 53
Endpoint application payload: 1125
Endpoint application payload: 36
User application payload: 36
```

Old browser flow

```
User application payload: 2220
Endpoint application payload: 98
Endpoint application payload: 362
Endpoint application payload: 41
User application payload: 41
User application payload: 2105
Endpoint application payload: 46
Endpoint application payload: 1330
Endpoint application payload: 41
User application payload: 41
User application payload: 2205
Endpoint application payload: 237
Endpoint application payload: 418
Endpoint application payload: 41

User application payload: 2220
User application payload: 41
Endpoint application payload: 98
Endpoint application payload: 259
Endpoint application payload: 41
User application payload: 41
User application payload: 2105
Endpoint application payload: 46
Endpoint application payload: 1306
Endpoint application payload: 41
User application payload: 41
User application payload: 2205
Endpoint application payload: 236
Endpoint application payload: 424
Endpoint application payload: 41
User application payload: 41
```

Newer browser flow

Statistical methods
(Block ciphers)

- We issue a large amount of requests for each item of the attack vector.
- We calculate the mean response length for each item.
- The correct guess should converge to smaller mean response length, compared to the others.

Statistical methods
(Huffman fixed-point)

- Huffman tables may be tampered, when different requests are issued.
- We describe a methodology to bypass this Huffman-induced noise:
  - An alphabet pool is created, containing every item in the alphabet of the secret.
  - In each request, the part of the alphabet that is not being tested is appended in the beginning.
  - Each request presents same letter frequency, although the text is rearranged.

Statistical methods
(Huffman fixed-point)

?q=rynmkwi_1_2_3_4_5_6_7_8_9_Credit Card: 0znq
?q=rynmkwi_0_2_3_4_5_6_7_8_9_Credit Card: 1znq
?q=rynmkwi_0_1_3_4_5_6_7_8_9_Credit Card: 2znq
?q=rynmkwi_0_1_2_4_5_6_7_8_9_Credit Card: 3znq
?q=rynmkwi_0_1_2_3_5_6_7_8_9_Credit Card: 4znq
?q=rynmkwi_0_1_2_3_4_6_7_8_9_Credit Card: 5znq
?q=rynmkwi_0_1_2_3_4_5_7_8_9_Credit Card: 6znq
?q=rynmkwi_0_1_2_3_4_5_6_8_9_Credit Card: 7znq
?q=rynmkwi_0_1_2_3_4_5_6_7_9_Credit Card: 8znq
?q=rynmkwi_0_1_2_3_4_5_6_7_8_Credit Card: 9znq

Statistical methods
(Hill-climbing parallelization)

- The alphabet partitioning follows a divide-and-conquer scheme.
- Example:
  - The attack vector on digits could be as follows:

    [“0 2 4 6 8”, “1 3 5 7 9”]
  - The correct digit will be compressed with the secret, so the vector item that contains it will present better behavior.
  - Each stage of the attack outputs a chosen half of the tested alphabet fragment, until the chosen half contains only one digit, which is the correct one.
- This method could reduce the time of the attack from O(|S|) to O(log|S|).

Statistical methods

(Cross-domain parallelization)

- Most websites use subdomains for specific applications, such as mobile versions.
- Cookies from the parent domain are available to the subdomains.
- If the subdomains handle similar data, containing the chosen secret, the attack could be issued against them.
- The parallelization could effectively increase the attack efficiency up to *N*x, where *N* is the number of different subdomains.

Statistical methods

(Point-system meta-predictor)

- Experiments revealed that the correct guess does not always result in minimum mean response length.
- However, the correct item is more probable to be among the *best* ones over time, compared to the others, that may demonstrate only a spike in performance for a certain period.
- For that reason we introduce a point-system that evaluates the performance of each item compared to the others.

1: 20	**2:** 16
3: 12	**4:** 10
5: 8	**6:** 6
7: 4	**8:** 3
9: 2	**10:** 1

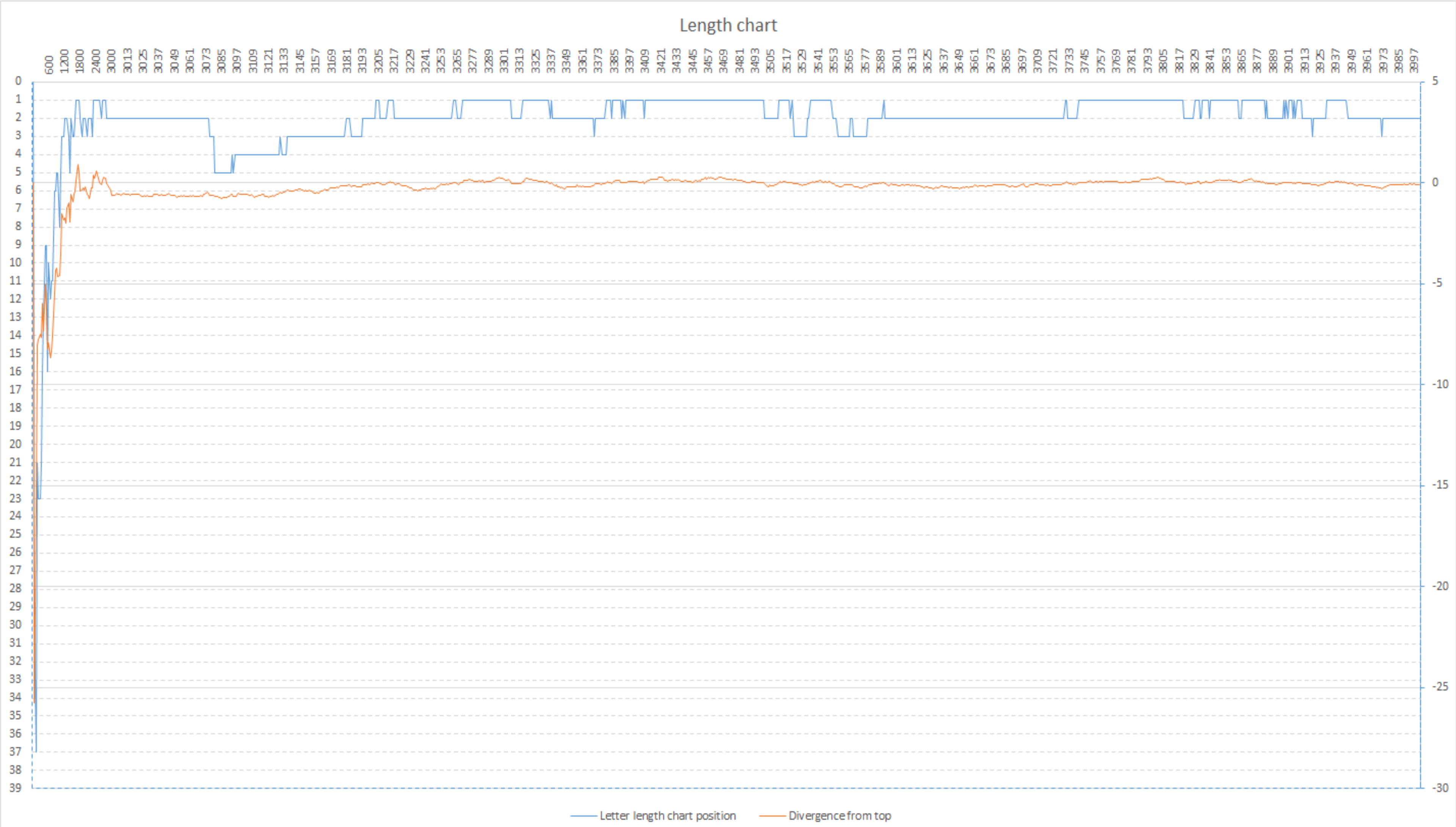
Experimental results
(Facebook Chat messages)

- We created a lab account, that has no friends, no user activity of any kind, except for a self-sent private message, containing the secret.
- We choose a prefix to bootstrap the attack, while the alphabet consists of lowercase and uppercase letters.
- We issue the attack using the serial method of requests, performing 4000 iterations, with a 4 second interval between requests.

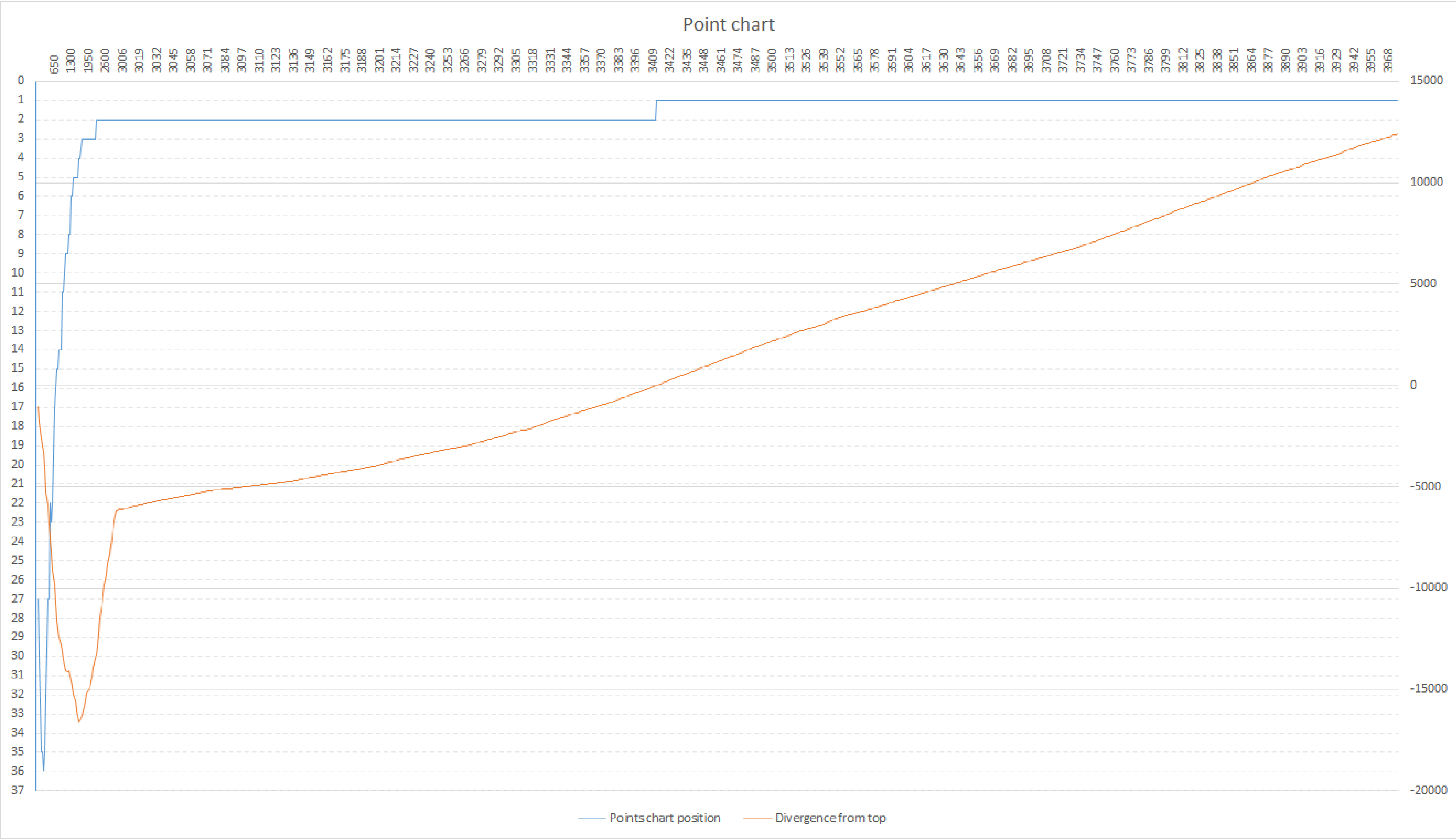
<u>Total time</u>

4000*52*4 = 832000 seconds = 9 days

Experimental results
(Facebook Chat messages)



Experimental results
(Facebook Chat messages)



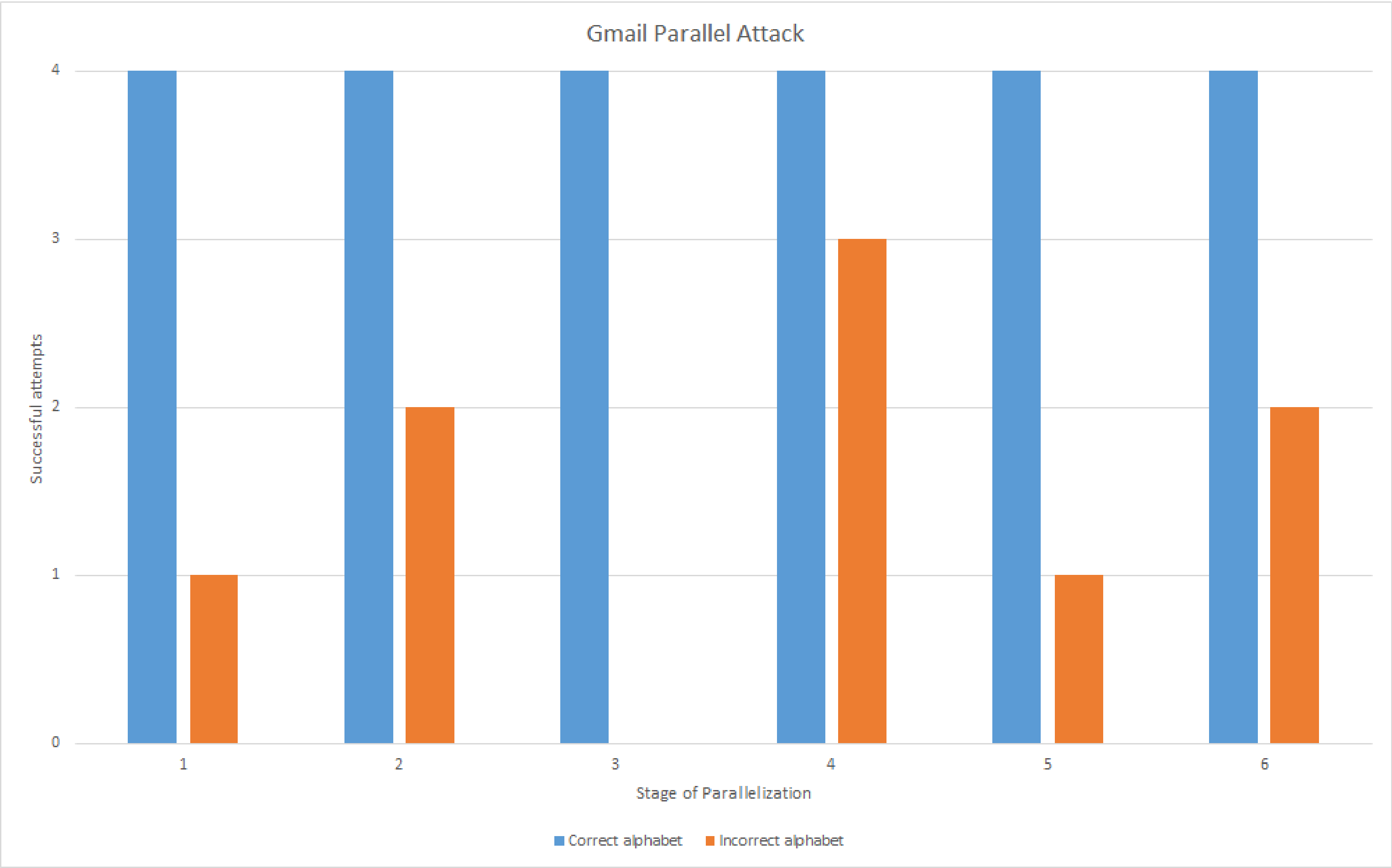
Experimental results
(Gmail Authentication token)

- We use the hill-climbing parallelized attack method to steal the auth token of a regular Gmail account.
- The alphabet consists of lowercase, uppercase, digits and dashes, so the stages of the attack are log(64) = 6.
- We repeat each stage of the attack, until one of the two halves is chosen 4 times, so at most 7 attempts are made for each stage of the parallelization.

<u>Total time</u>

4000*7*6*4 = 672000 seconds = 7 days

Experimental results
(Gmail Authentication token)



Mitigation techniques

- [Prado etc.] proposed several mitigation techniques:
  - **Length hiding**. In this work, we were able to defeat this mitigation measure through noise by-passing.
  - **Separating secrets from user input**. In this work, we were able to defeat this mitigation measure through alternative secrets: Secrets and user input are sometimes one and the same, e.g. private messages.
  - **Masking secrets**. This mitigation mechanism is still feasible. But we showed that many more secrets than CSRF tokens must be masked.
  - **Rate limiting and monitoring**. This mitigation mechanism is still feasible.
  - **CSRF protection**. In this work, we showed that this is not adequate mitigation, as secrets other than CSRF can be stolen.
  - **Disabling compression**. While this solves the problem, it is not a practical solution.

Novel mitigation techniques
(Compressibility annotation)

- We propose that web servers and web application servers cooperate to indicate which portions must not be compressed.
- Web application server returns annotated response:
  - Annotation indicates where secrets are located.
  - Annotation indicates where reflection is located.
  - Annotation uses some special format.
- Must be implemented separately in every web framework, e.g. Django, Ruby on Rails.
- Web server interprets annotated web application server response and changes compression behavior.
- Annotated reflections and secrets always sent as literals
- Must be implemented separately in web servers, e.g. mod_breach for Apache, Nginx etc.

Novel mitigation techniques (SOS headers)

- [Schema, Toukharian ‘13] propose SOS headers as an extension to CSP.
- A policy applies to each cookie, specifying whether it should be included in a request.
- Policies applied: any, self, isolate
- Pre-flight requests are made to check for exceptions.
- If trusted websites use HSTS policy and cookies are not included in other cases, the response would not contain the secret.
- Complete mitigation of the attack.

Conclusion

- Our contributions:
  - Definition of IND-PCPA
  - Attack optimization:
    - Parallelization
    - Point-system prediction
    - Attack persistence
  - Alternative secrets
  - Experimental results on major systems
- Future work:
  - Mathematical proof for IND-PCPA properties
  - HTTP injection persistency mechanism
  - Integration of MitM attacks
  - Implementation of proxy on TCP level
  - Implementation of novel mitigation techniques

Thank you!

Questions?