A Perfect CRIME? TIME Will Tell

Tal Be'ery, Imperva

OWASP
The Open Web Application Security Project

Presenter: Tal Be'ery, CISSP

OWASP
The Open Web Application Security Project

- Web Security Research Team Leader at Imperva
- Holds MSc & BSc degree in CS/EE from TAU
- 10+ years of experience in IS domain
- Facebook "white hat"
- Speaker at RSA, BlackHat, AusCERT
- Columnist for securityweek.com

Agenda

OWASP
The Open Web Application Security Project

- Introduction
  - Compression Primer
  - CRIME attack revisited
- Expanding CRIME
  - Increasing the attack surface with HTTP responses
- TIME attack
  - Exploiting timing side channel
- Conclusions & mitigations

OWASP
The Open Web Application Security Project

Introduction

Compression on the Web
Pre-CRIME

OWASP
The Open Web Application Security Project

- Based on the GZIP algorithm
- Common compression
  - HTTP Response Body
- Uncommon compressions
  - HTTP Request body
  - Header compression
  - SSL/TLS Compression
    - Servers: Open SSL, others
    - Clients: Chrome
  - SPDY
    - Servers: Apache MOD_SPDY, others
    - Clients: All but IE

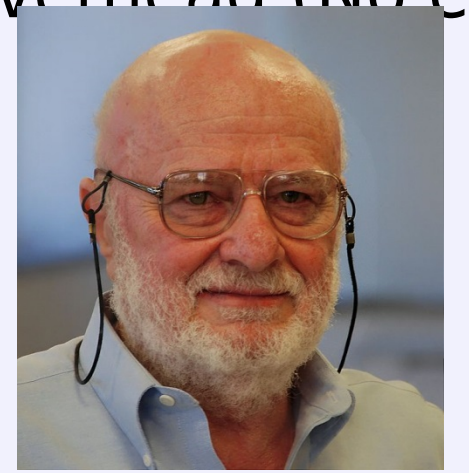
GZIP/DEFLATE Compression

OWASP
The Open Web Application Security Project

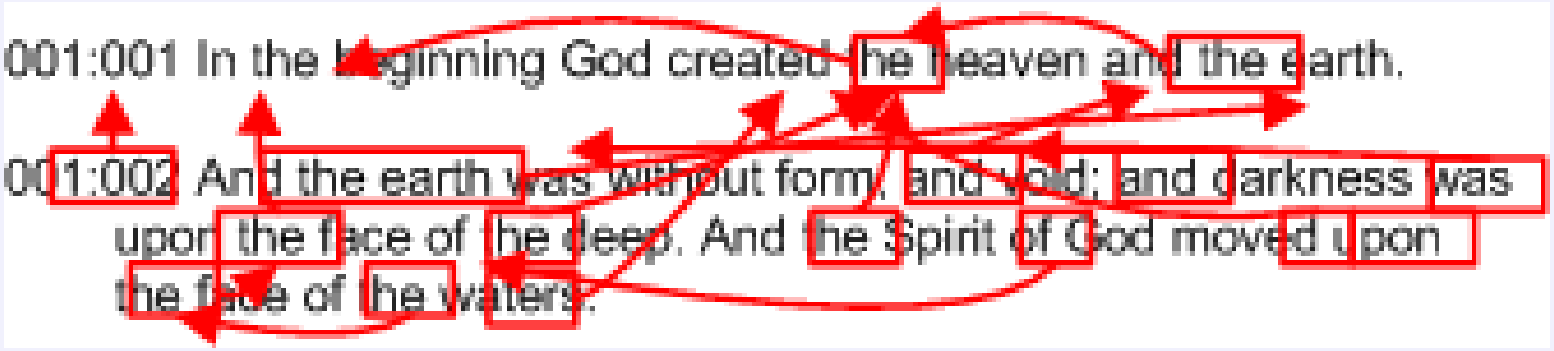
- Two step compression process
  - LZ77 to compress reoccurring strings
  - Huffman code to compress frequent symbols
- Good compression rate with low overhead
  - Memory
  - CPU
  - Compression dictionaries

Compression – LZ Algorithms

- Lempel Ziv, late 70s
- Compress repeating strings
  - Lossless
  - Asymptotically optimal
  - No overhead (No extra dictionary)



LZ Compression – Example

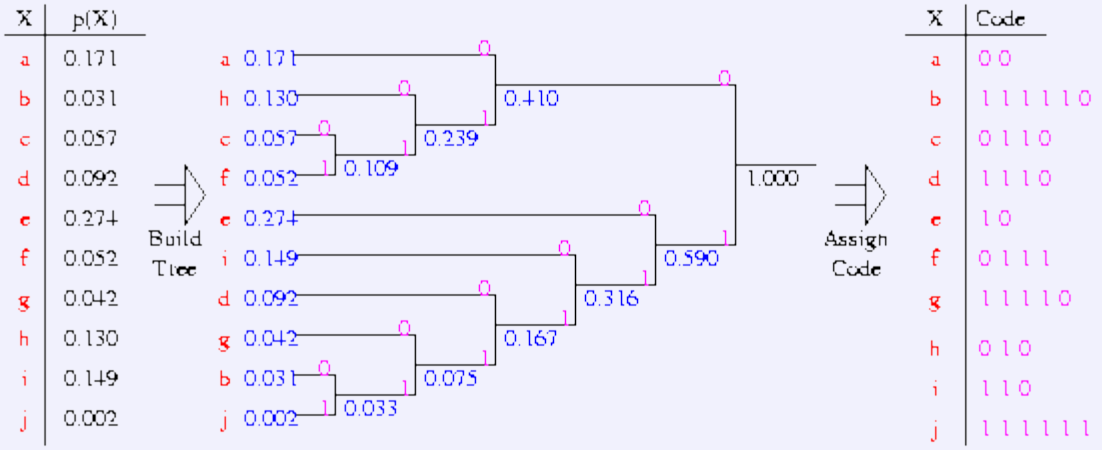


- 001:001 In the beginning God created<25, 5>heaven an<14, 6>earth. 0<63, 5>2 A<23, 12> was without form,<55, 5>void;<9, 5>darkness<40, 4> <0, 7>upo<132, 6>face of<11, 5>deep.<93, 9>Spirit<27, 4><158, 4>mov<156, 3><54, 4><67, 9><62, 16>w<191, 3>rs

Huffman Code

OWASP
The Open Web Application Security Project

- David Huffman - 1952
- Assign shorter codes (in bits) for frequent symbols (letters)



Compression & Encryption

OWASP

The Open W ion Security Project

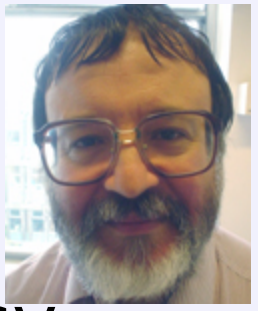
?

Compression & Encryption

!

Compression leaks data!

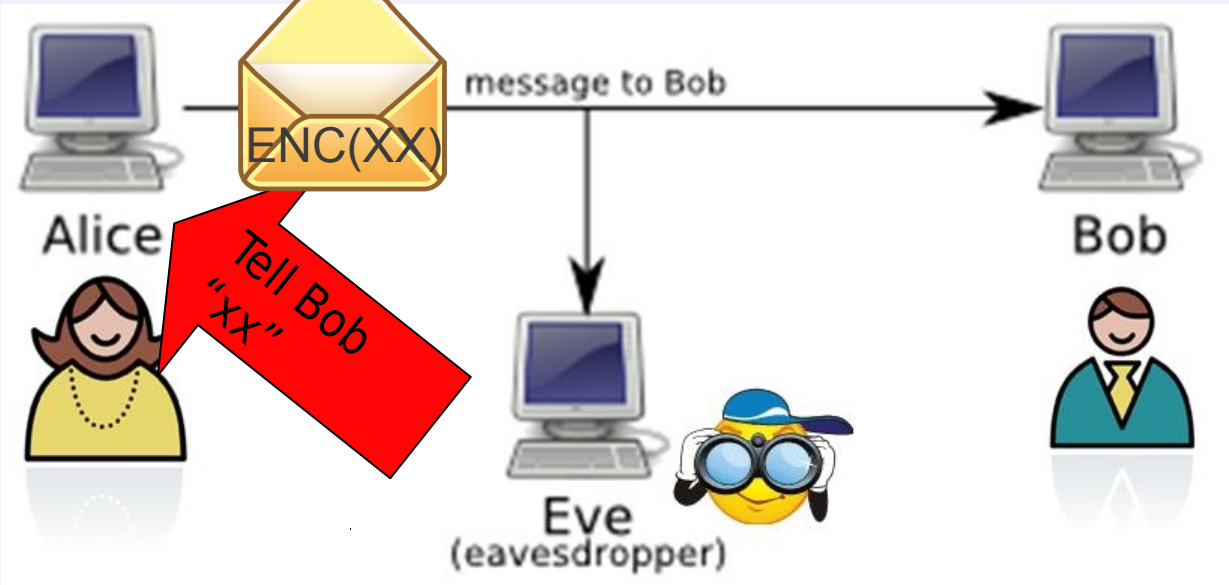
- Kelsey - 2002 : “Compression and Information Leakage of Plaintext”
- Rizzo and Duong – 2012
- Compression Ratio Info-leak Made Easy (CRIME)
- Chosen Plaintext Attack
- Targets compression informat



Chosen Plaintext Attack Model

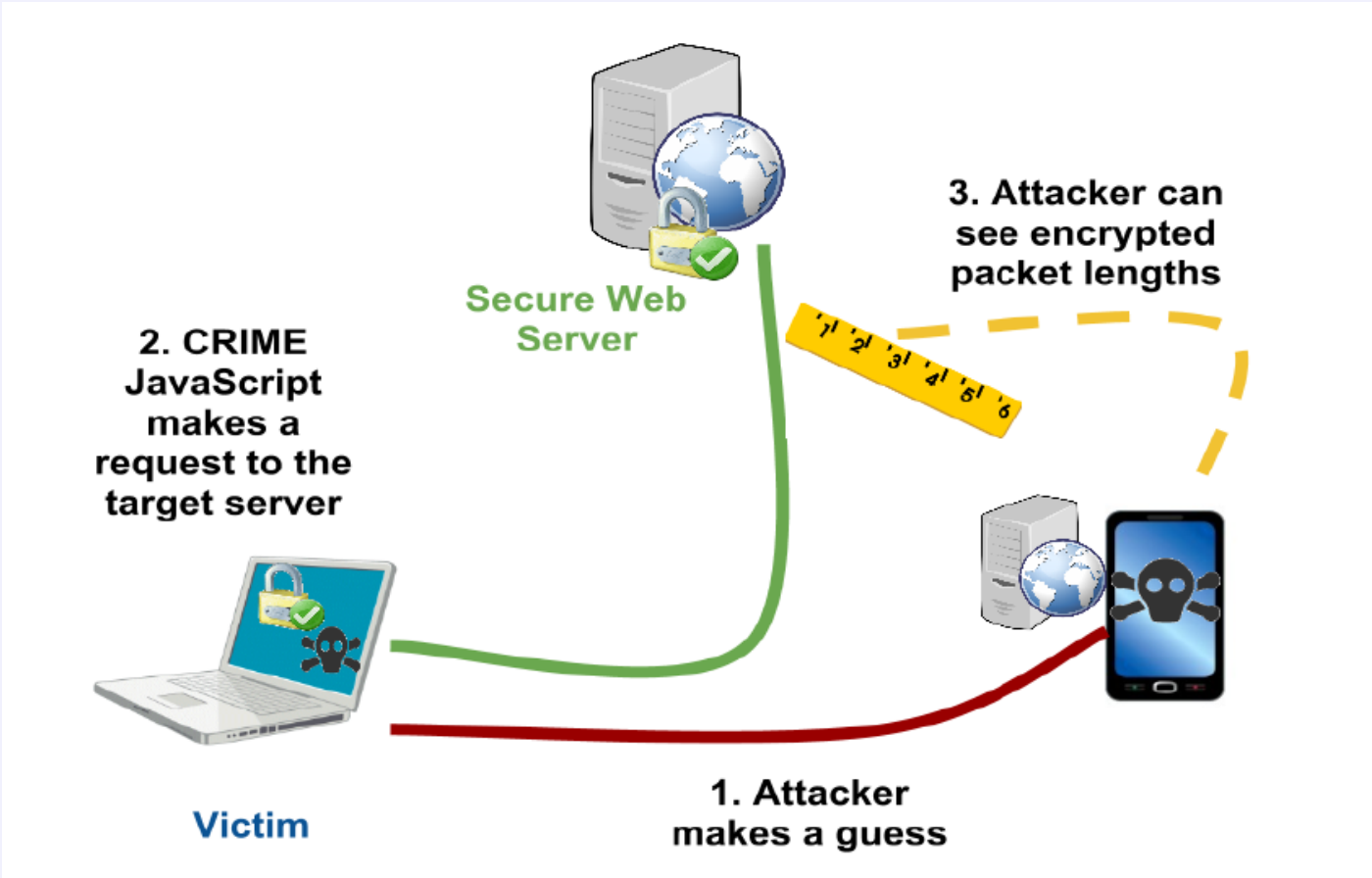
OWASP
The Open Web Application Security Project

- **A chosen-plaintext attack (CPA)** is an attack model for cryptanalysis which presumes that the attacker has the capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding ciphertexts.



CRIME in a Slide

OWASP
The Open Web Application Security Project



oung & Rizzo original presentation
https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/present#slide=id.g1d134dff_1_157

Under CPA Model Compression Leaks Data

OWASP
The Open Web Application Security Project

- CPA attacker algorithm:
  - Guess(0) = a known prefix of the secret string
  - Symbol = array of the secret alphabet (i.e {a,b,c..})
  - Until the whole secret is recovered
  - Guess(n) = Guess(n-1) + symbol(i)
  - Payload = original payload + Guess(n)
  - Measure length
    - For a correct guess String repeated gets compressed shorter length (encryption does not change size)
  - If successful
    - n++,i=0 // proceed to guessing the next secret's symbol
  - Else
    - i++ // try another alphabet symbol
  - Repeat

CPA and the Web

OWASP
The Open Web Application Security Project

- Attacker is an eavesdropper – can see ciphered text
- Attacker creates HTTP request interactively (via script)
  - Full control (almost): URL
  - Can predict: Most headers
  - Does not control or see: cookies
  - Encrypted on wire
  - Not accessible from script
    - Same Origin Policy
    - “HTTP only”

Compression Leaks Data

OWASP
The Open Web Application Security Project

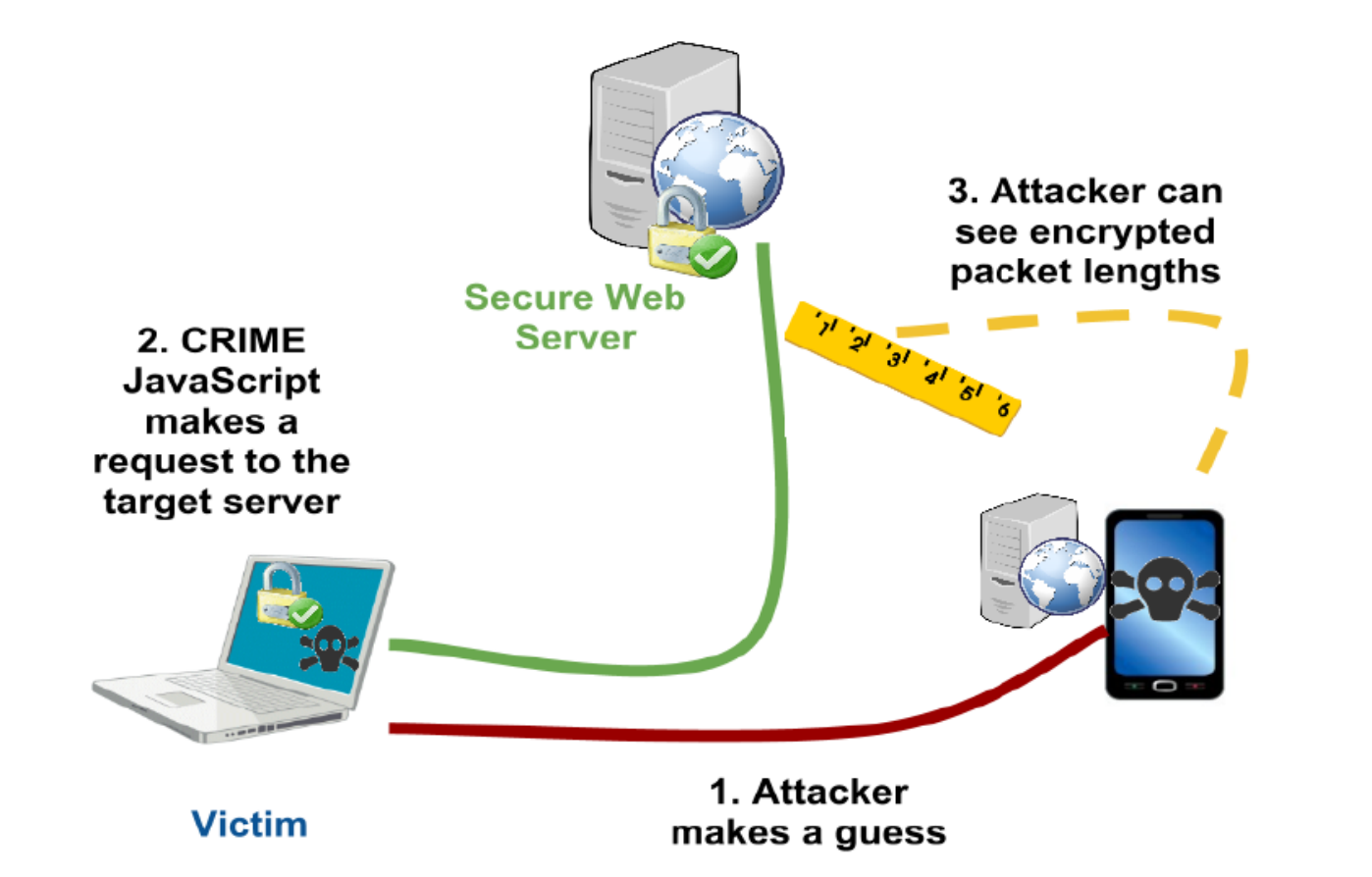
- Attack model
  - Use the URL attacker controls
  - Guess byte by byte

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

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

CRIME in a Slide

OWASP
The Open Web Application Security Project



Doung & Rizzo original presentation
https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/present#slide=id.g1d134dff_1_157

Practical Issues

OWASP
The Open Web Application Security Project

- Some issues with Huffman coding
  - Some chars representation < 1 byte
  - Good guess might get unnoticed
- Solution
  - Send some more requests with the same chars blend (same Huffman coding) in different order to (different LZ77 compression) to eliminate chars redundancy issues

CRIME Aftermath

OWASP
The Open Web Application Security Project

- SPDY implementations cancel/modify header compression
- Chrome disabled SSL compression

Resurrecting CRIME

OWASP
The Open Web Application Security Project

Extending CRIME for HTTP responses

HTTP Response

OWASP
The Open Web Application Security Project

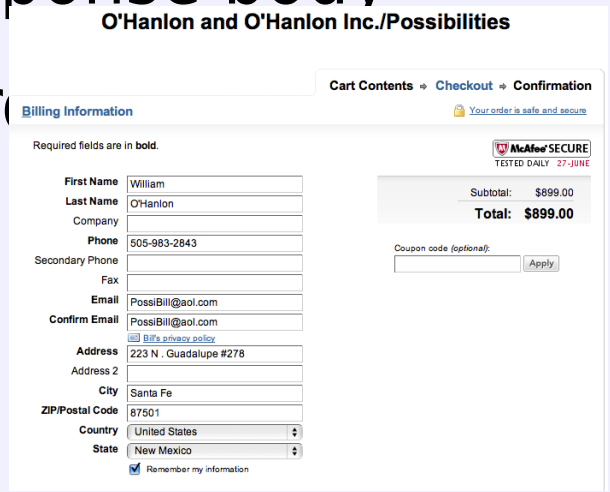
- CRIME attack “ingredients”

CRIME element	HTTP request	HTTP response
Encryption	SSL	SSL
Compression	GZIP	GZIP
Secret element location	Request header	Response body
Secret element	Cookie value	Application specific
Secret element prefix/suffix	Cookie name	Application specific
Chosen plain text location	URL	Application specific

Secrets in Response Data

OWASP
The Open Web Application Security Project

- We need a secret with a known prefix/suffix
- Luckily, they are everywhere..
  - The applications' secrets are in their content i.e. delivered by HTTP response body
  - Secrets are often structur fixed prefix or suffix



HTTP Response

OWASP
The Open Web Application Security Project

- CRIME attack “ingredients”

CRIME element	HTTP request	HTTP response
Encryption	SSL	SSL
Compression	GZIP	GZIP
Secret element location	Request header	Response body
Secret element	Cookie value	Application specific
Secret element prefix/suffix	Cookie name	Application specific
Chosen plain text location	URL	Application specific

Embedding Chosen Text

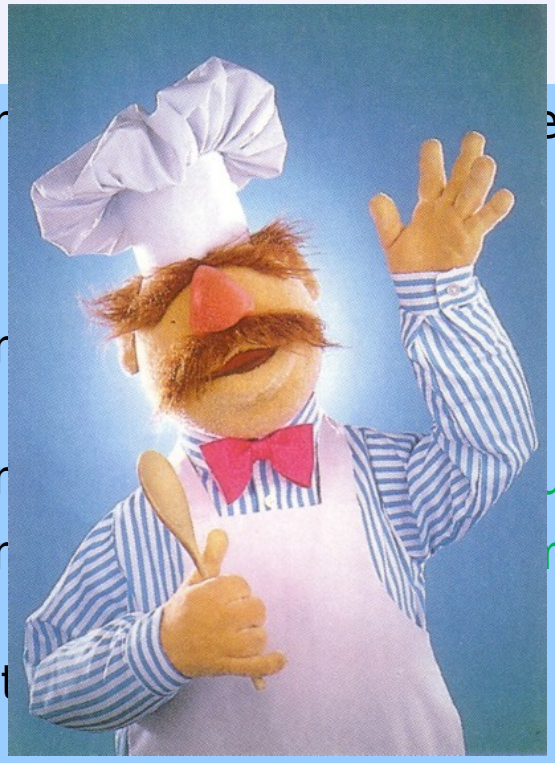
OWASP
The Open Web Application Security Project

- Application specific - yet not infrequent
- Many applications embeds user input (as expressed with HTTP parameters) within their response
- In fact, many times parameters will be embedded even if there are no parameters on the original request

HTTP Response

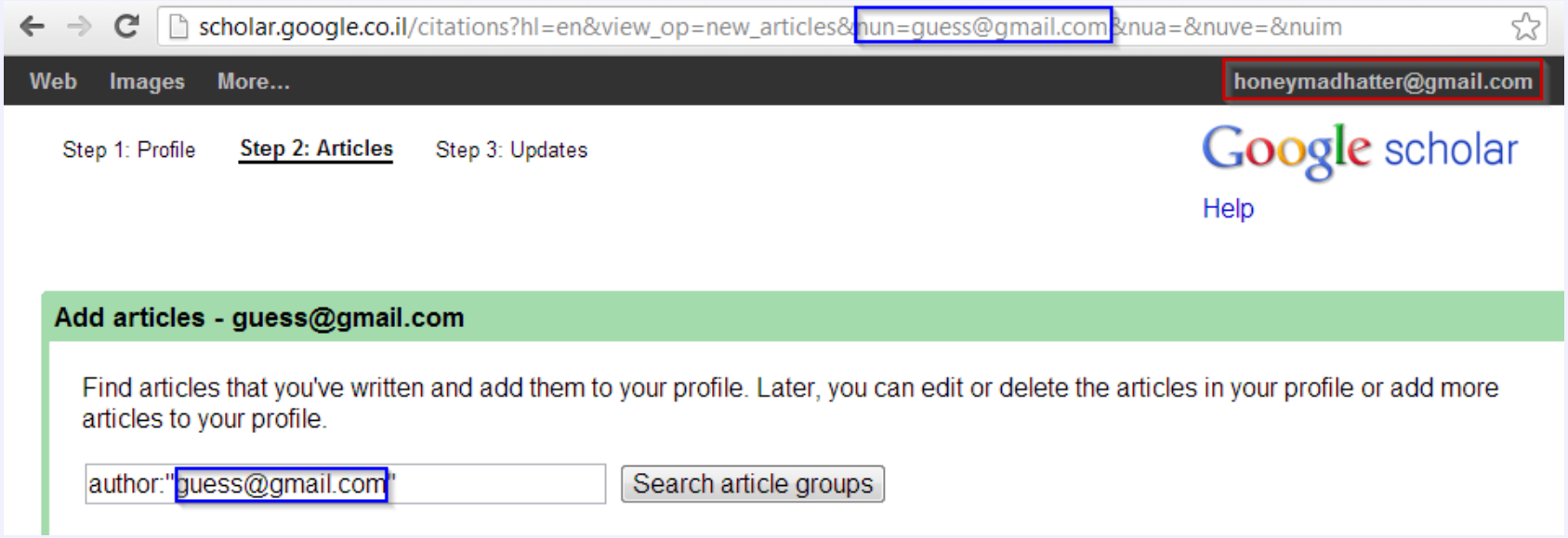
OWASP
The Open Web Application Security Project

- CRIME attack “ingredients”

CRIME elemer		est HTTP response
Encryption		SSL
Compression		GZIP
Secret elemer		Response body
Secret elemer		ue Application specific
Secret elemer prefix/suffix		me Application specific
Chosen plain t		Application specific

Google Scholar PoC

OWASP
The Open Web Application Security Project



Demo

Response vs. Request Pros

OWASP
The Open Web Application Security Project

- HTTP response body compression is a very common practice - cannot be easily turned off
- Attacking the secret data itself and not some intermediate (cookie)

Response vs. Request Cons

OWASP
The Open Web Application Security Project

- User input is encoded before embedding into the response to protect against injection attacks. Therefore the attack target is limited to mostly alphanumeric characters.
- Less sterile environment:
  - Response body might be altered due to other reasons
  - Input might get embedded more than once

The TIME attack

Motivation

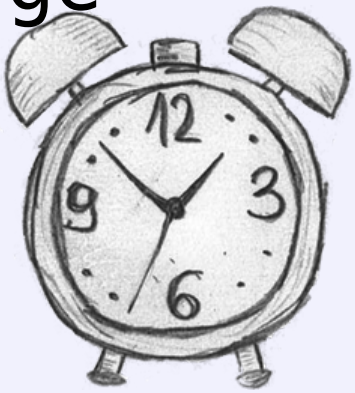
OWASP
The Open Web Application Security Project

- Crime attack model has some very limiting attack preconditions: Eavesdropping AND web page control
- Directing user traffic to a controlled site is a fairly easy task
- But eavesdropping to victim's traffic with other site is a much harder requirement
- If only we could drop the eavesdropping requirement…

TIME

OWASP
The Open Web Application Security Project

- Imperva – 3.2013
- Timing Info-leak Made Easy (TIME)
- Chosen Plaintext Attack
- Targets timing information leakage



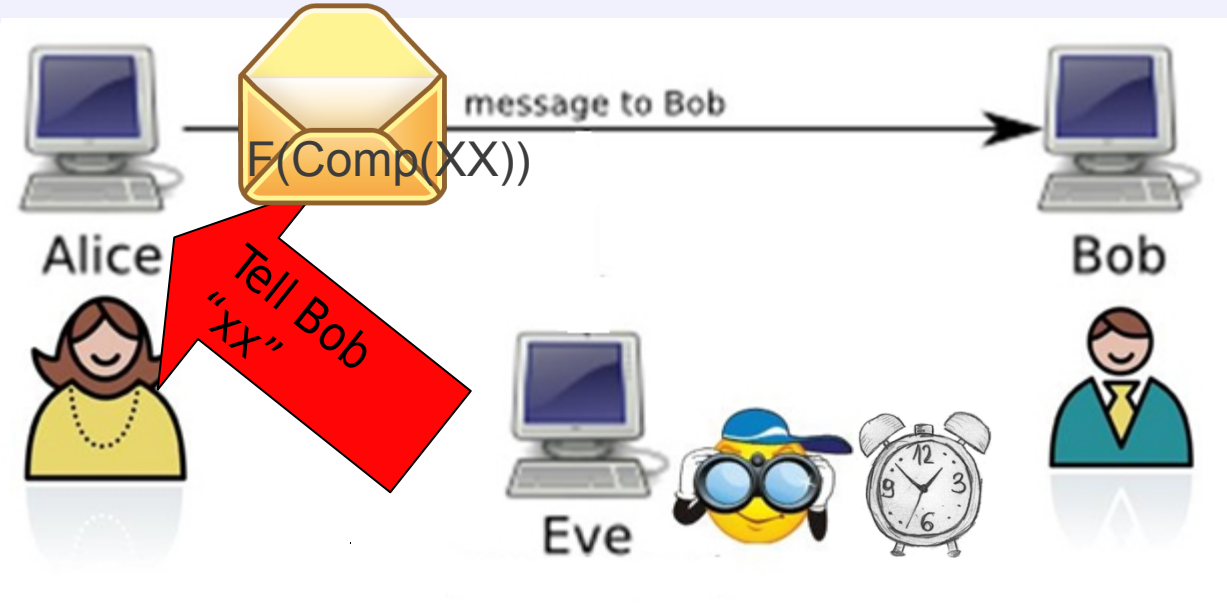
TIME Argument Outline

OWASP
The Open Web Application Security Project

- HTTP Payload size may carry sensitive information
  - Moreover, HTTP payload size **differences** detection is sufficient to extract the sensitive information
- Using timing measurements attacker can distinguish HTTP payload size **differences**
- These timing measurements can be done with javascript on attacker site
- **Result – attackers can learn the user's sensitive information using javascript from their site, with no eavesdropping!**

Attack Model

- Attacker has the capability to choose arbitrary plaintexts and obtain timing observations on their traffic
- **Attacker no longer needs to be an eavesdropper!**
  - **Expanding the attack scope**



Sensitive Info in HTTP Payload Size

OWASP
The Open Web Application Security Project

- HTTP request
  - CRIME for request to extract cookie data
- HTTP response
  - Extended CRIME to extract response data
  - Access a behind authentication resource for user login status detection
  - Application specific: e.g. number of digits in bank account balance
- Moreover, HTTP payload size **differences** detection is sufficient to extract the sensitive information

User Login Status Detection

OWASP
The Open Web Application Security Project

I Know What Websites You Are Logged-In To (Login-Detection via CSRF)

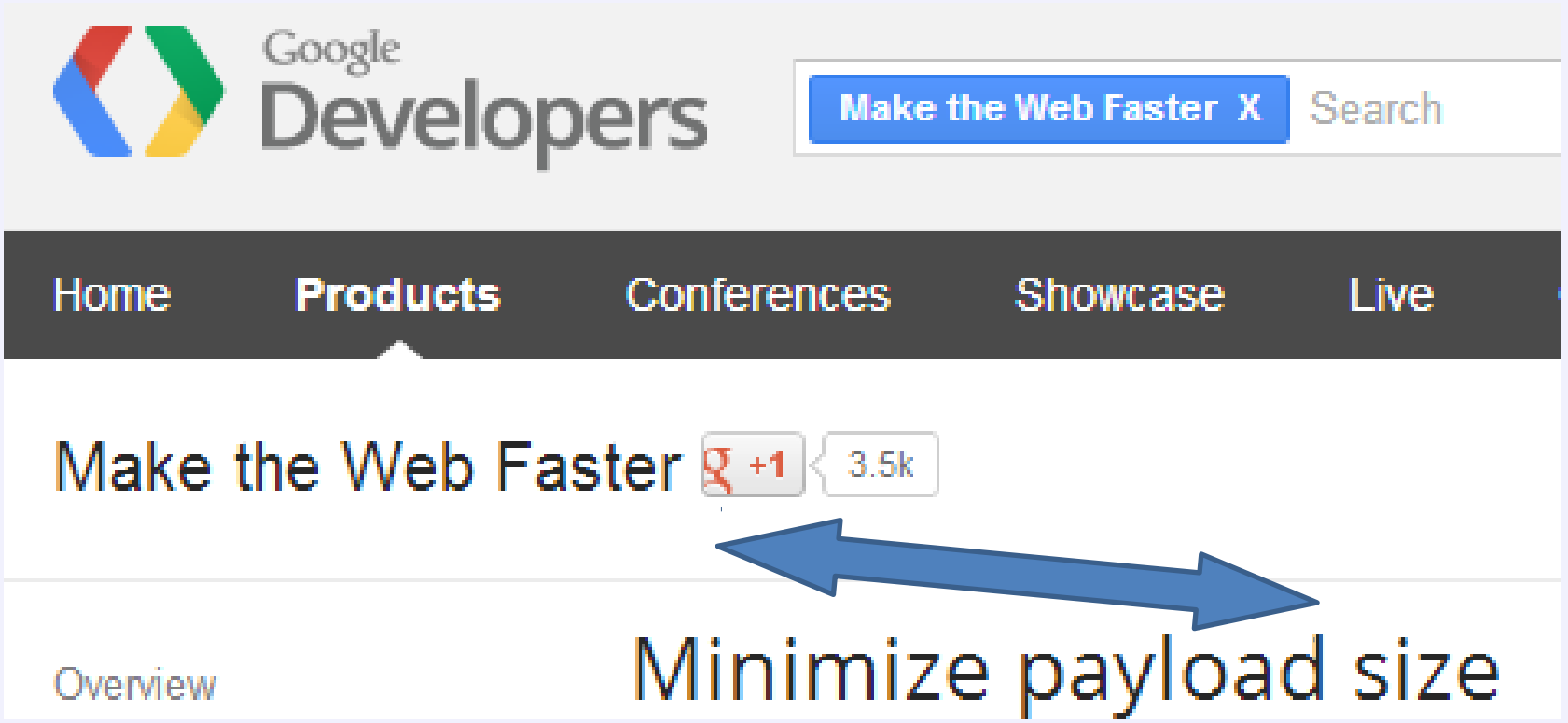
OCTOBER 10, 2012 BY JEREMIAH GROSSMAN 3 COMMENTS

Now that we know a lot about a visitor's browser, we can mix and match several techniques – six, by my count — that http://maliciouswebsite/ can use to learn what other websites a visiting browser is logged in to – an online bank, social network, email provider, a local home router's Web interface, and basically anything else.

18 Tweet

Timing Reveals Payload Size Diff-1

OWASP
The Open Web Application Security Project



https://developers.google.com/speed/docs/best-practices/payload

Timing Reveals Payload Size Diff-2

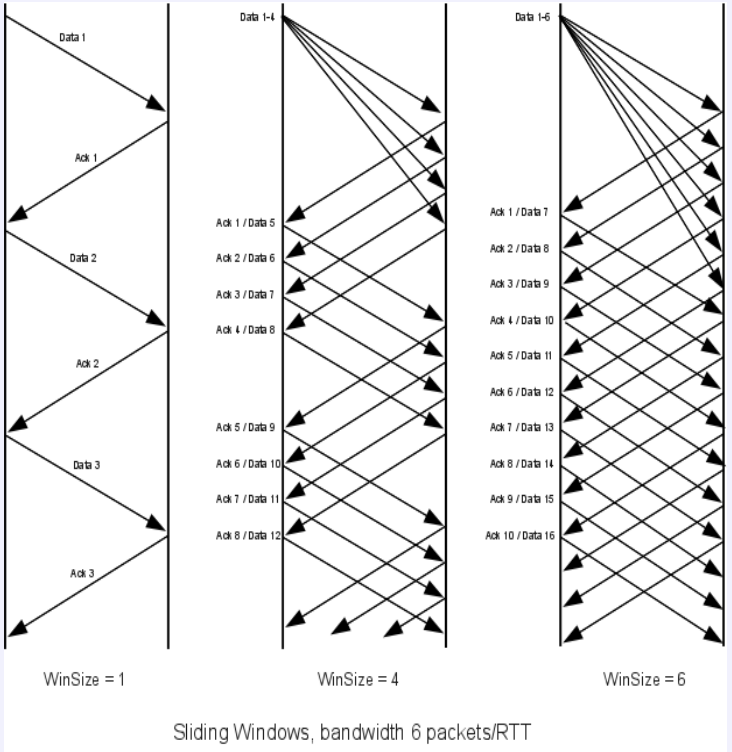
OWASP
The Open Web Application Security Project

- Google web page speed tips for developers:
  - “The amount of data sent in each server response can add significant latency to your application”
  - “In addition to the network cost of the actual bytes transmitted, there is also a penalty incurred for crossing an IP packet boundary. “

https://developers.google.com/speed/docs/best-practices/payload

Timing Oracle

- Client send a window of packets
- Waits RTT for ACK
- RTT time is noticeable
- attacker can easily distinguish
  - Size(request) <= window
  - Size(request) > window
- If payload length is exactly on data boundary, attacker can determine 1 byte differences



http://ulam2.cs.luc.edu/ebook/chap03.html

Request Timing

OWASP
The Open Web Application Security Project

- Sent with Chrome
- Sends 2 packets and wait
- If you need to send 3 packets – pay extra

No.	Time	Protocol	Length	Info
2284	0.000000000	TCP	66	27983 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 W
2298	0.177681000	TCP	66	http > 27983 [SYN, ACK] Seq=0 Ack=1 Win=14480 Len=
2299	0.000092000	TCP	54	27983 > http [ACK] Seq=1 Ack=1 Win=65536 Len=0
2317	0.183176000	TCP	1514	[TCP segment of a reassembled PDU]
2318	0.000016000	TCP	1514	[TCP segment of a reassembled PDU]
2326	0.169969000	TCP	60	http > 27983 [ACK] Seq=1 Ack=1461 Win=8960 Len=0
2327	0.000052000	HTTP	55	GET /?FTYnCuZg9XheUnuABl7mM9aUGk7XtutuTdxsybNa9imA
2328	0.000039000	TCP	60	http > 27983 [ACK] Seq=1 Ack=2921 Win=11776 Len=0
2332	0.167268000	TCP	60	http > 27983 [ACK] Seq=1 Ack=2922 Win=11776 Len=0
2333	0.006509000	TCP	1502	[TCP segment of a reassembled PDU]

Response Timing

OWASP
The Open Web Application Security Project

- This Apache server implements a window of 3 packets
- If it needs to send the fourth – pays extra RTT

734	0.001188000	10.2.141.22	66.147.244.96	HTTP	504 GET /wp-content/uploads/2012/09/criminal1.jpg?y=389513423&uiouo=348975390 HTTP/1.1
735	0.184815000	66.147.244.96	10.2.141.22	TCP	60 http > 52135 [ACK] Seq=1 Ack=451 Win=6912 Len=0
736	0.003336000	66.147.244.96	10.2.141.22	TCP	1514 [TCP segment of a reassembled PDU]
737	0.000308000	66.147.244.96	10.2.141.22	TCP	1514 [TCP segment of a reassembled PDU]
738	0.000078000	10.2.141.22	66.147.244.96	TCP	54 52135 > http [ACK] Seq=451 Ack=2921 Win=65536 Len=0
739	0.000027000	66.147.244.96	10.2.141.22	TCP	1514 [TCP segment of a reassembled PDU]
740	0.182835000	66.147.244.96	10.2.141.22	TCP	1514 [TCP segment of a reassembled PDU]
741	0.000173000	10.2.141.22	66.147.244.96	TCP	54 52135 > http [ACK] Seq=451 Ack=5841 Win=65536 Len=0
742	0.000036000	66.147.244.96	10.2.141.22	TCP	1514 [TCP segment of a reassembled PDU]
743	0.000063000	66.147.244.96	10.2.141.22	TCP	1514 [TCP segment of a reassembled PDU]
744	0.000058000	10.2.141.22	66.147.244.96	TCP	54 52135 > http [ACK] Seq=451 Ack=8761 Win=65536 Len=0
746	0.182404000	66.147.244.96	10.2.141.22	TCP	1514 [TCP segment of a reassembled PDU]

HTTP Request's Timing with JS

OWASP
The Open Web Application Security Project

- Create HTTP request with XHR
  - XHR adheres to SOP
  - Allows GET requests to flow
  - If headers allow show response
  - If not, abort
  - We don't care for the response
  - Timing leaks the request size
- Use getTime() on XHR events
  - onreadystatechange
- Noise elimination
  - Repeat the process (say 10 times) and obtain **Minimal** time

HTTP Request's Timing with JS
PoC Setup

OWASP
The Open Web Application Security Project

- HTML with Javascript, sending method is XHR
- PoC target edition.cnn.com
- Sends one byte diff requests alternately 10 times
- The longer request crosses the send window boundary
- The shorter is exactly within

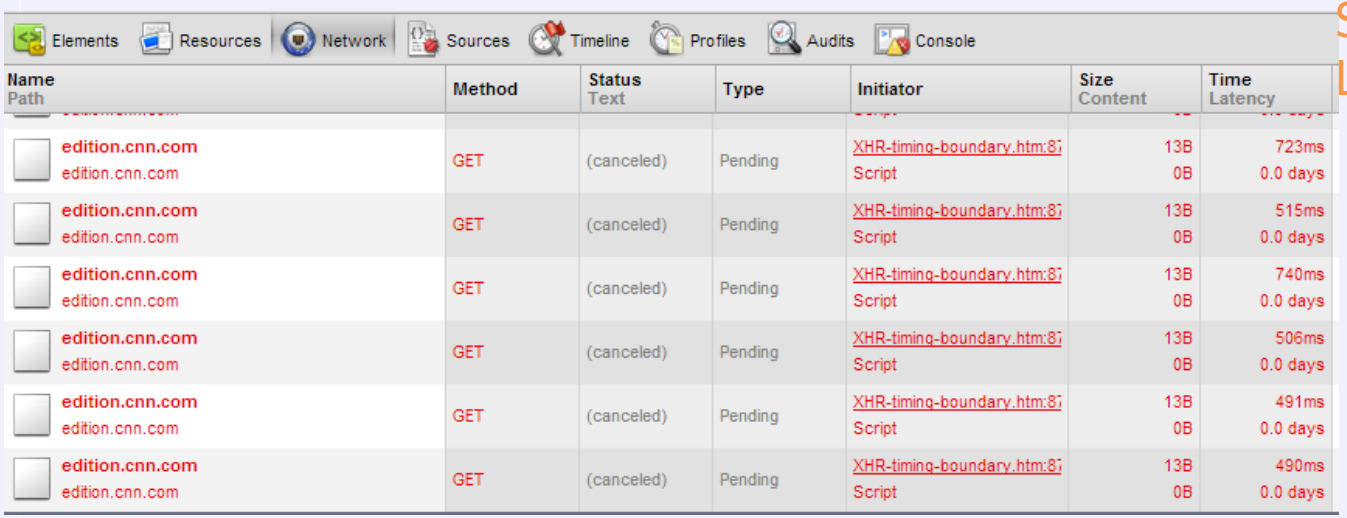
Script results

- Measures requests time
- Outputs length and time
- Outputs the minimal timing values for both requests' length

HTTP Request's Timing with JS PoC Results

OWASP
The Open Web Application Security Project

- Timing can be correctly captured
- Results are conclusive



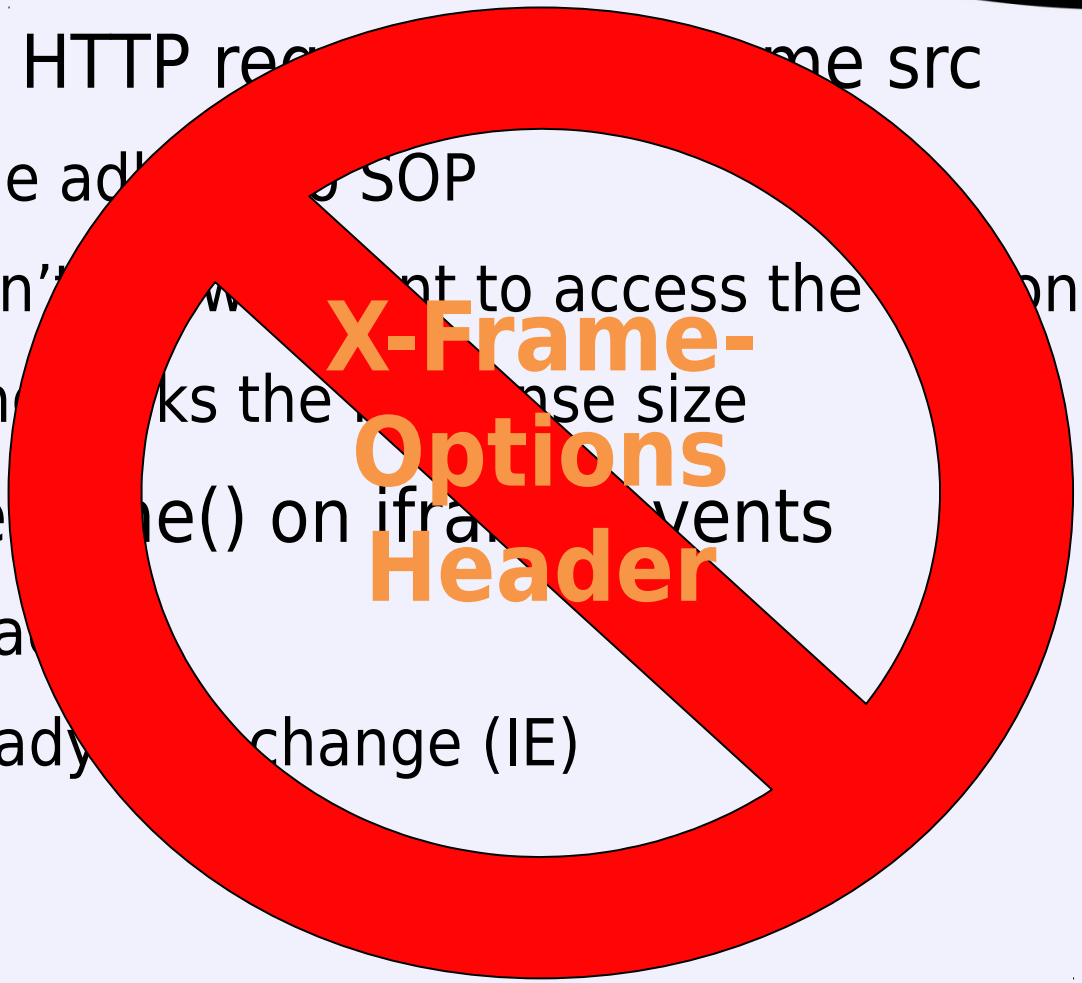
Script results
Length,Time

2515,717
2514,512
2515,738
2514,504
2515,490
2514,490
Min first 468
Min Second 246

HTTP Response Timing with JS Take 1

OWASP
The Open Web Application Security Project

- Create HTTP req me src
  - iframe ad SOP
  - Doesn' nt to access the onse content
  - Timin ks the nse size
- Use ge e() on ifra vents
  - onLoa
  - Onready change (IE)



HTTP Response Timing with JS Take 2

OWASP
The Open Web Application Security Project

- Create HTTP request with IMG src
  - Target resource is fetched even if not an image
  - not tamed by the X-Frame-Options header
  - Timing leaks the response size
- Use getTime() on img events
  - onLoad
  - Onreadystatechange (IE)

TIME Argument Outline - Revisit

OWASP
The Open Web Application Security Project

- HTTP Payload size may carry sensitive information
  - Moreover, HTTP payload size **differences** detection is sufficient to extract the sensitive information
- Using timing measurements attacker can distinguish HTTP payload size differences
- These measurements can be done with javascript on attacker site
- **Result – attackers can learn the user's sensitive information using javascript from their site, with no eavesdropping!**

MISSION: ACCOMPLISHED

Where was SOP?

OWASP
The Open Web Application Security Project

- SOP = Same Origin Policy
- “SOP - a mechanism that governs the ability for JavaScript and other scripting languages to access DOM properties and methods across domains”
- In order to prevent malicious scripts served from the attacker site to leak data from other site – browsers apply the Same Origin Policy (SOP)

https://code.google.com/p/browsersec/wiki/Part2

The Greater Lesson: Automation Introduces New Risks

OWASP
The Open Web Application Security Project

- Simple multimedia tags are exempt from SOP
  - Fetching images from other domains is OK
- Enabling the Img src manipulation by a javascript does not seem to change the model
- However, due to automation, it does
  - Interactively setting the URL
  - Measuring load time
- Breaks SOP – allow data leak from one domain to another

OWASP
The Open Web Application Security Project

Conclusions & mitigations

Key Contributions

OWASP
The Open Web Application Security Project

- Resurrecting the CRIME attack with Extended CRIME attack against responses
- Introduced TIME attack to launch size diff attacks with **no eavesdropping requirement**
  - Original CRIME
  - Extended CRIME
  - Login detection
  - Application specific – e.g. # of digits in bank balance

Non-Mitigations

- Add Random Time Delays
- Lucky thirteen authors: “A natural reaction to timing based attacks is to add random time delays... to frustrate statistical analysis. In fact, this countermeasure is surprisingly ineffective”

Browsers

OWASP
The Open Web Application Security Project

- Browser should support and respect “X-Frame-Options” header for all content inclusion (not just IFRAME)
- By thus, allowing applications to take control over the presentation of their content on other domains

Applications

OWASP
The Open Web Application Security Project

- Take control over your content
  - Implement CSRF protection
  - Use the X eader
- Beware of neters
- Deploy an easures

Questions

Compression infoleak attacks Timeline

OWASP
The Open Web Application Security Project

- 2002: Scientific paper, Kelsey
  - “Compression and Information Leakage of Plaintext”
- 2012: CRIME, Rizzo+Duong
  - Actual exploit for HTTP requests
- 3.2013: TIME, Be’ery + Shulman
  - Actual exploit for HTTP responses
  - Trading MITM with timing inference
- 7.2013: BREACH, Gluck+Harris+Prado
  - Actual exploit for HTTP responses