Practical New Developments in the BREACH Attack

Dimitris Karakostas
Dionysis Zindros
HTTPS is **broken**

- BREACH broke HTTPS + RC4 in 2013
- People upgraded to AES – thought they were safe

Today...

- We show TLS + AES is **still broken**
- **HTTPS can be decrypted** - quick and easy
- We launch **open source tool** to do it here in Singapore
Overview

• BREACH review
• Our contributions
• Statistical attacks
• Attacking block ciphers
• Attacking noise
• Optimization techniques
• Our tool: Rupture
• Mitigation recommendations
Original BREACH research

Introduced in Black Hat USA 2013

Angelo Prado    Neal Harris    Yoel Gluck
BREACH attack anatomy
Original BREACH assumptions

Target website:

- Uses **HTTPS**
- Compresses response using **gzip**
- Uses **stream cipher**
- Response has **zero** noise
- Contains end-point that **reflects** URL parameter
Original BREACH target

1. Steal **secret** in HTTPS response (CSRF tokens)
2. Use CSRF to impersonate victim client to victim server
Length leaks

\[ |E(A)| < |E(B)| \iff |A| < |B| \]
Let’s attack Gmail

- **m.gmail.com** mobile Gmail view
- Mobile search functionality uses HTTP POST – but HTTP GET still works :)
- CSRF token included in response – valid for all of Gmail
<base href="https://mail.google.com/mail/u/0/x/pugg7ui43zaf-/"><input type="hidden" name="nredir" value="?&amp;q=blackhatblackhat&amp;"><input type="hidden" name="search" value="query"><div class="noMatches">No results for: blackhatblackhat</div><script type="text/javascript">var token="AF6bupMJX-9CU4zxp362SDbN49o45nMjSg";var searchPageLinks=document.getElementsByClassName("searchPageLink");for(i=0;i<searchPageLinks.length;i++)searchPageLinks[i].oncl
• Attacker guesses part of secret
• Uses it in reflection
• Compressed/encrypted response is shorter if right!
Original BREACH methodology

- Guess part of secret and insert into reflection
- Match? → **Shorter** length due to compression
- No match? → **Longer** length
- Bootstrap by guessing 3-byte sequence
- Extend **one character** at a time
- $O(n|\Sigma|)$ complexity
  - $n$: length of secret
  - $\Sigma$: alphabet of secret
Can we really attack Gmail?

- Uses AES
- Has random bytes in response
Our contributions
Our contributions

We extend the BREACH attack

1. Attack *noisy* end-points
2. Attack *block cipher* end-points
3. **Optimize** attack
4. Propose novel mitigation techniques

The *whole web* is vulnerable
Statistical methods
Statistical methods

- We can attack noisy end-points
- Multiple requests per alphabet symbol
- Take mean response length
- m-sized noise → attack works in $O(n|\Sigma|\sqrt{m})$
  - $m = (\text{max response size}) - (\text{min response size})$
- Length converges to correct results (LLN)
Statistical methods against block ciphers

- Everyone uses block ciphers
- Statistical methods break them
- We introduce artificial noise
- Block ciphers round length to 128-bits
- In practice **16x more requests**
- Blocks aligned $\mapsto$ Length difference measurable
Experimental results

- AES_128 is vulnerable
- Popular web services are vulnerable:
  - Gmail
  - Facebook
  - etc.
Optimizations
Optimizations overview

Block ciphers cause 16x slowdown. We need to optimize.

- **Divide and conquer**: 6x speed-up
- **Request soup**: 16x speed-up
- **Browser parallelization**: 6x speed-up

Total ~ 500x speed-up!
Optimization: Divide & Conquer

- Each request tries multiple candidates from alphabet
- Partition alphabet using divide-and-conquer
- Binary search on alphabet partitions
- Reduces attack complexity from $O(n|\Sigma|)$ to $O(n \lg|\Sigma|)$
- Practically this gives 6x speed-up
Binary search in alphabet space
Optimization: Request soup

Problem:

- Need 16x samples for block ciphers
- But we only need the length mean

Solution:

- Responses come pipelined, can’t tell them apart
- We don’t care! Measure total length
- Divide by amount, extract mean
Optimization: Browser parallelization

- Do 6x parallel requests; browsers support it
- Each parallel request cannot adapt based on previous
- But we need many samples of same candidates anyway
- No need to adapt before we collect enough
Today, we make BREACH easy

- Over the past months, we’ve developed **rupture**
- Today in Black Hat Asia 2016, we make it **open source**

[https://github.com/dionyziz/rupture](https://github.com/dionyziz/rupture)

ruptureit.com
Rupture

• Extensible
  • Modular analysis / optimizations / strategies
  • Experiment with your own
• General web attack framework
  • Can be adapted to work for CRIME, POODLE, …
  • Persistent command & control channel
• Scalable architecture: Multiple attacks simultaneously
• Come help us make it better
RUPTURE ARCHITECTURE

Amazon
Ebay
CNN
DeviantArt

Victim
Injector

Victim Network

Real-Time
Node.js
Service

HTTP
JS

HSTS
HTTPS

Target

HTTP

Strategy
Analyst

Backend
Django

Adversary

Adversary Network

SQLite DB
Robust, persistent command & control

- Automatically inject JS to HTTP
- All plaintext connections infected
- One tab at a time gets work from C&C server
- User closes tab? Different tab starts attacking
- User switches browsers? Works on different browser
- Data collection failed for a sample? Sample recollected
- User reboots computer? Attack continues
Persistent attack data storage

- Collected data processed by Django middleware
- Attack historical data **stored permanently** in MySQL db
- Future analysis with new techniques possible
Rupture demo
Mitigation
First-party cookies

- Don’t send auth cookies cross-origin
- Backwards compatibility: Web server opts-in
- Mike West implemented it in Chrome 51
- Coming April 8th

Set-Cookie: SID=31d4d96e407aad42; First-Party
Key takeaways

1. HTTPS + gzip = **broken**
2. Rupture framework is live – **attacks are easy**
3. Enable **first-party cookies** on your web app
Thank you! Questions?

twitter.com/dionyziz
45DC 00AE FDDF 5D5C B988 EC86 2DA4 50F3 AFB0 46C7

github.com/dimkarakostas
DF46 7AFF 3398 BB31 CEA7 1E77 F896 1969 A339 D2E9