Compression Bombs Strike Back

Giancarlo Pellegrino
gpellegrino@mmci.uni-saarland.de

BeNeLux OWASP Day 2016
November 25\textsuperscript{th}, Leuven, Belgium
About Me

- Post doctoral researcher of the System Security group at CISPA, Saarland University, Germany
- Research focus:
  - Web application security / security protocols
  - Vulnerability detection (logic vulns, Server-Side Requests Abuses, CSRF)
- Former member of S3 group at EURECOM, Sophia-Antipolis, France
- Former research associate in the Security & Trust research group at SAP SE
Modern applications rely on (core) network services, e.g., Web, email, and IM services.
Introduction

- Modern applications rely on (core) network services, e.g., Web, email, and IM services
- Amount of exchanged data continues to increase steadily
  - More data → more transfer time → unresponsiveness → user unhappiness
Introduction

- Modern applications rely on (core) network services, e.g., Web, email, and IM services
- Amount of exchanged data continues to increase steadily
  - More data → more transfer time → unresponsiveness → user unhappiness
  - Avg web page size as Doom ~2.3MB [1]

Introduction

- Modern applications rely on (core) network services, e.g., Web, email, and IM services
- Amount of exchanged data continues to increase steadily
  - More data → more transfer time → unresponsiveness → user unhappiness
- Solution 1: buy more bandwidth!
Introduction

- Modern applications rely on (core) network services, e.g., Web, email, and IM services
- Amount of exchanged data continues to increase steadily
  - More data → more transfer time → unresponsiveness → user unhappiness
- Solution 1: buy more bandwidth!
  ➔ Bandwidth costs
Introduction

- Modern applications rely on (core) network services, e.g., Web, email, and IM services

- Amount of exchanged data continues to increase steadily
  - More data → more transfer time → unresponsiveness → user unhappiness

- Solution 1: buy more bandwidth!
  - Bandwidth costs

- Another solution is …
Modern applications rely on core network services, e.g., web, email, and IM services. The amount of exchanged data continues to increase steadily.

- More data → more transfer time → unresponsiveness → user unhappiness
- Solution 1: buy more bandwidth!
  - Bandwidth costs
- Another solution is ...
Data Compression

- Reduces # of bits of a string by removing redundancy
  - *lossless* if $\text{decompr}(\text{compr}(d)) = d$ or *lossy* if $\text{decompr}(\text{compr}(d)) \neq d$

- Lots of algorithms (See [1])

- Among the most popular: Deflate [RFC 1951]
  - Implemented in libraries, e.g., zlib, or as a tool, e.g., gzip, and zip archive tool
  - Available in most of the programming languages

Compression in Protocols

- Compression used by network protocols to reduce message size
- Mandated by protocol specifications
  - e.g., HTTP (response!) compression, IMAP, XMPP, SSH, PPP, and others
- Or implemented as custom feature
  - e.g., HTTP request compression

- IMAP Compression [RFC 4978]
- XMPP Compression [XEP-0138]
- HTTP Compression [RFC 7230]
Compression in HTTP (RFC 7230)

HTTP Request
GET / HTTP/1.1
Host: wikipedia.org
[...]
Compression in HTTP (RFC 7230)

HTTP Request
GET / HTTP/1.1
Host: wikipedia.org
[...]

HTTP Response
HTTP/1.1 200 OK
[...]
Content-Length: 82170
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html><html
[...]

~80Kb of page

Retrieve default HTML page
Compression in HTTP (RFC 7230)

HTTP Request
GET / HTTP/1.1
Host: wikipedia.org
Accept-Encoding: gzip, deflate
[...]

November 28, 2016
Compression in HTTP (RFC 7230)

HTTP Request
GET / HTTP/1.1
Host: wikipedia.org
Accept-Encoding: gzip, deflate

HTTP Response
HTTP/1.1 200 OK
Content-Length: 18879
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip

Response size -70%
Compressed response body
Decompress
Select algorithm
The Problem of Data Compression

- If not properly implemented, it can make application vulnerable to DoS

- Risks:
  1) **Intensive task**
     - Computationally intensive
     - If abused, it can stall an application
  2) **Data Amplification**
     - Decompression increases the data to be processed (compression rate of zlib ~ 1:1024)
     - Internal components may not be designed to handle high volume of data
  3) **Unbalanced Client-Server Scenario**
     - One party pre-compute compressed messages
     - The other one decompresses messages each time

- Popular examples from the past...

- 42 KB zip file → **4.5 PB** uncompressed data

- 5 layers of nested zip files in blocks of 16, last layer with text files of 4.3 GB each

- Cause Disk/Memory exhaustion

- Sent as attachment to crash anti-virus software

- Resource exhaustion in libxml2 when processing nested XML entity definitions

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ELEMENT lolz (#PCDATA)>
  <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;"> 
  <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;"> 
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;"> 
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;"> 
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;"> 
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;"> 
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;"> 
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;"> 
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;"> 
]> 
<lolz>&lol9;</lolz>
```

- 810 bytes of XML document expanded to 3GB
The Past: Zip Bombs and Billion Laughs

This was 1996-2003!
Now we know better, right?
The Present

- Reviewed protocol specs, design patterns, and coding rules

**Unawareness** of the risks, **guidelines** on handling data compression are **missing** or **misleading**
The Present

- Reviewed protocol specs, design patterns, and coding rules

  **Unawareness** of the risks, **guidelines** on handling data compression are **missing** or **misleading**

1. Protocol specifications:
   - No data compression handling issues, redirects to SSL/TLS (concerned with leakage and packet limits, but unexplained how they apply to other protocols)
The Present

- Reviewed protocol specs, design patterns, and coding rules
  
  Unawareness of the risks, guidelines on handling data compression are missing or misleading

1. Protocol specifications:
   - No data compression handling issues, redirects to SSL/TLS (concerned with leakage and packet limits, but unexplained how they apply to other protocols)

2. Secure Design Patterns:
   - Patterns to solve vulns. during design phase: DoS Safety, Compartmentalization, and Small Process
   - However, lack of the details to address implementation-level concerns
The Present

- Reviewed protocol specs, design patterns, and coding rules

  **Unawareness** of the risks, **guidelines** on handling data compression are **missing** or **misleading**

1. **Protocol specifications:**
   - No data compression handling issues, redirects to SSL/TLS (concerned with leakage and packet limits, but unexplained how they apply to other protocols)

2. **Secure Design Patterns:**
   - Patterns to solve vulns. during design phase: *DoS Safety, Compartmentalization, and Small Process*
   - However, lack of the details to address implementation-level concerns

3. **Secure Coding Rules**
   - Only one, i.e., Anti-Zip Bomb coding rule
   - Sadly, incorrect
The Present

- Reviewed protocol specs, design patterns, and coding rules

Unawareness of the risks, guidelines on handling data compression are missing or misleading

1. Protocol Specifications
   - No data compression handling issues, redirects to SSL/TLS (concerned with leakage and packet limits, but unexplained how they apply to other protocols)

2. Secure Design Patterns
   - Patterns to solve vulns: During design phase: DoS Safety, Compartmentalization, and Small Process
   - However, lack of the details to address implementation level concerns

3. Secure Coding Rules
   - Only one, i.e., Anti-Zip Bomb coding rule
   - Sadly, incorrect

How does this lack of common knowledge and understanding affect implementations?
Impact on Implementations
HTTP (Response) Compression (RFC 7230)

HTTP Request
GET / HTTP/1.1
Host: wikipedia.org
Accept-Encoding: gzip, deflate

HTTP Response
HTTP/1.1 200 OK
[...]
Content-Length: 18879
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip

Decompress

Response size -70%

Compressed response body

Select algorithm
Compression Bombs against Web Browsers #1

HTTP Request
GET / HTTP/1.1
Host: attacker.foo
Accept-Encoding: gzip, deflate

HTTP Response
HTTP/1.1 200 OK
Content-Length: 4000000
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip

4 GB of white spaces
Compression rate ~1:1000

See: Geoff Jones http://blog.cyberis.co.uk/2013/08/vulnerabilities-that-just-wont-die.html
Compression Bombs against Web Browsers #1

HTTP Request
GET / HTTP/1.1
Host: attacker.foo
Accept-Encoding: gzip, deflate

HTTP Response
HTTP/1.1 200 OK
Content-Length: 4000000
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip

Compression rate ~1:1000
4 GB of white spaces

See: Geoff Jones http://blog.cyberis.co.uk/2013/08/vulnerabilities-that-just-wont-die.html

November 28, 2016
Compression Bombs against Web Browsers #2

HTTP Request
GET / HTTP/1.1
Host: attacker.foo
Accept-Encoding: gzip, deflate

HTTP Response
HTTP/1.1 200 OK
Content-Length: 4000
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip, gzip

See: Geoff Jones http://blog.cyberis.co.uk/2013/08/vulnerabilities-that-just-wont-die.html
Compression Bombs against Web Browsers #2

HTTP Request
GET / HTTP/1.1
Host: attacker.foo
Accept-Encoding: gzip, deflate

HTTP Response
HTTP/1.1 200 OK
Content-Length: 4000
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip, gzip

See: Geoff Jones http://blog.cyberis.co.uk/2013/08/vulnerabilities-that-just-wont-die.html
HTTP (Response) Compression Bombs

"Vulnerabilities that just won't die - Compression Bombs"
by Geoff Jones
http://blog.cyberis.co.uk/2013/08/vulnerabilities-that-just-wont-die.html

<table>
<thead>
<tr>
<th></th>
<th>Internet Explorer</th>
<th>Firefox</th>
<th>Chrome/Chromium</th>
<th>Safari</th>
<th>Opera</th>
</tr>
</thead>
<tbody>
<tr>
<td>1TBx4 HTML</td>
<td>Not supported</td>
<td>See 3</td>
<td>See 6</td>
<td>Not supported</td>
<td>See 10</td>
</tr>
<tr>
<td>1TB HTML</td>
<td>See 1</td>
<td>See 3</td>
<td>See 6</td>
<td>See 9</td>
<td>See 11</td>
</tr>
<tr>
<td>1TBx4 FILE</td>
<td>Not supported</td>
<td>See 4</td>
<td>See 7</td>
<td>Not supported</td>
<td>See 12</td>
</tr>
<tr>
<td>1TB FILE</td>
<td>See 2</td>
<td>See 5</td>
<td>See 7</td>
<td>See 9</td>
<td>See 12</td>
</tr>
<tr>
<td>1TB SDCH</td>
<td>Not supported</td>
<td>Not supported</td>
<td>See 8</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

Most are still vulnerable!
How about servers?
Experiments

- Case studies:
  - HTTP, XMPP, and IMAP servers

- Testbed:

  ![Diagram showing Internal Monitor, Implementation, External Monitor, Attackers, and Compression bombs connected with arrows.]

  Internal Monitor → Implementation
  Implementation → External Monitor
  Internal Monitor → /proc
  /proc → Linux 3.8 Kernel
  Attackers → Compression bombs
HTTP (request) Compression Bomb (SOAP)

- Case studies:
  - HTTP, XMPP, and IMAP servers
- Testbed:
  - Internal Monitor Implementation
  - Linux 3.8 Kernel /proc
  - External monitor by Attackers

Compression bombs
- ~4 MB, ~1:1000 compr. ratio

Example HTTP request:

```plaintext
POST /index.html HTTP/1.1
Content-Encoding: gzip
\r\n
<soapenv:Envelope>
  <soapenv:Body><![CDATA[ [...]]]></soapenv:Body>
</soapenv:Envelope>
\r\n```

Compressed

4 GB of white spaces

Same for JSON
XMLP Compression Bomb

- Case studies:
  - HTTP, XMPP, and IMAP servers

- Testbed:

```xml
<?xml version='1.0' ?>
<stream:stream to='server'
    xmlns='jabber:client' Version='1.0'>

  4 GB of white spaces

xmlns='jabber:client' Version='1.0'>
```

Compression bombs

~4 MB, ~1:1000 compr. ratio
IMAP Compression Bomb

- Case studies:
  - HTTP, XMPP, and IMAP servers
- Testbed:

```
From: sender@foo
To: receiver@foo
Subject: I am a bomb!
```

~4 MB, ~1:1000 compr. ratio

*4 GB of white spaces*
## Compression Bombs Everywhere

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Network Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>XMPP</td>
<td>OpenFire</td>
</tr>
<tr>
<td></td>
<td>Prosody</td>
</tr>
<tr>
<td></td>
<td>Tigase</td>
</tr>
<tr>
<td></td>
<td>Tigase, jabberd2</td>
</tr>
<tr>
<td>HTTP</td>
<td>Apache HTTPD + mod_deflate</td>
</tr>
<tr>
<td></td>
<td>+ mod-php, CSJRPC, mod-gsoap, mod-dav</td>
</tr>
<tr>
<td></td>
<td>Apache Tomcat + 2Way/Webutilities filter</td>
</tr>
<tr>
<td></td>
<td>+ Apache CXF</td>
</tr>
<tr>
<td></td>
<td>+ json-rpc, lib-json-rpc</td>
</tr>
<tr>
<td></td>
<td>+ Axis2/ +jsonrpc4j</td>
</tr>
<tr>
<td></td>
<td>Axis 2 standalone</td>
</tr>
<tr>
<td></td>
<td>gSOAP standalone</td>
</tr>
<tr>
<td>IMAP</td>
<td>Dovecot, Cyrus</td>
</tr>
</tbody>
</table>
## Compression Bombs Everywhere

### Protocol | Network Service
--- | ---
XMPP | OpenFire
| Prosody
| Tigase
| Ejabberd, jabberd2

**HTTP**

- Apache HTTPD + mod_deflate
  - + mod-php, CSJRPC, mod-gsoap, mod-dav
- Apache Tomcat + 2Way/Webutilities filter
  - + Apache CXF
  - + json-rpc, lib-json-rpc
  - + Axis2/ +jsonrpc4j

- Axis 2 standalone
- gSOAP standalone

- IMAP
  - Dovecot, Cyrus

### CVE-References

- CVE-2014-2741
- CVE-2014-2745
- CVE-2014-2746
- CVE-2014-0118
- Notif. devel
- Notif. devels
Pitfalls
Pitfalls

1. Implementation

2. Specification

3. Configuration
Pitfalls

1. Implementation
   - Use of Compression before Authentication
   - Improper Input Validation during Decompression
   - Logging Decompressed Messages
   - Improper Inter-Units Communication
   - Unbounded Resource Usage (CPU and Memory)

2. Specification
   - Erroneous Best Practice
   - Misleading Documentation
   - API Specs Inconsistency

3. Configuration
   - Insufficient Configuration Options
   - Insecure Default Values
   - Decentralized Configuration Parameters
Pitfalls

1. Implementation
   - Use of Compression before Authentication
   - Improper Input Validation during Decompression
   - Logging Decompressed Messages
   - Improper Inter-Units Communication
     - Unbounded Resource Usage (CPU and Memory)

2. Specification
   - Erroneous Best Practice
   - Misleading Documentation
   - API Specs Inconsistency

3. Configuration
   - Insufficient Configuration Options
   - Insecure Default Values
   - Decentralized Configuration Parameters
Pitfalls

1. Implementation
   - Use of Compression before Authentication
   - Improper Input Validation during Decompression
   - Logging Decompressed Messages
   - Improper Inter-Units Communication

3. Configuration
   - Insufficient Configuration Options
   - Insecure Default Values
   - Decentralized Configuration Parameters

Check out our paper!

http://trouge.net/gp/papers/compr_usenix15.pdf
Pitfalls at Implementation level

- Abstract message processing pipeline extracted from our case studies
Compression before Authentication

- Inconsistent best practice
  - Mandatory in SSL/TLS, recommended in XMPP, and undefined in IMAP and HTTP
  - Implementation may diverge from the specs, i.e., OpenSSH
- Developers may underestimate the risk or overlook recommendations
- Prosody accepted compressed messages before user authentication
  ➔ DoS by unauthenticated attackers
3 ways to validate a message:

- Compressed message size
  - mod-deflate: If (compr. size > LimitRequestBody) → Reject

  -> However, hard to assess message size from its compressed form (1 MB compr → 1 GB decompr.)
Improper Input Validation during Decompression

- 3 ways to validate a message:
  - Compressed message size
    - mod-deflate: If (compr. size > LimitRequestBody) → Reject
      - However, hard to assess message size from its compressed form (1 MB compr → 1 GB decompr.)
  - Decompression ratio
    - Patched mod-deflate: if (decompr. ratio > threshold) → Reject
      - Problem of ratio selection
Improper Input Validation during Decompression

3 ways to validate a message:

- Compressed message size
  - mod-deflate: If (compr. size > LimitRequestBody) → Reject
  - However, hard to assess message size from its compressed form (1 MB compr → 1 GB decompr.)

- Decompression ratio
  - Patched mod-deflate: if (decompr ratio > threshold) → Reject
  - Problem of ratio selection

- Decompressed message size
  - mod-deflate + mod-dav: If (decompr. size > LimitXMLRequestBody) → Reject

CVE-2014-0118
Improper Inter-Units Communication

- Upon exception, the pipeline halts and rejects message
- mod-php and mod-gsoap limit the size of incoming (decompressed) message
- ... but had no means to halt mod-deflate
  - mod-deflate keeps on decompressing data
  - Problem addressed in CVE-2014-0118
Logging Decompressed Messages

- Frequency and verbosity of log events can cause DoS
- If exception is caused by compressed data, the needed resources may be underestimated
- Upon invalid requests, Apache CXF logs first 100KB of incoming message
  - However, first it decompresses the entire message on a file, then logs the first 100KB
  - DoS due to disk space exhaustion
  

CVE-2014-0109/ -0110
Erroneous Best Practices (Spec. level)

- Only one code pattern specific for data compression
  - Rule: “IDS04-J. Safely extract files from ZipInputStream”
Erroneous Best Practices (Spec. level)

- Only one code pattern specific for data compression
  - Rule: “IDS04-J. Safely extract files from ZipInputStream”

```java
// Write the files to the disk, but
// only if the file is not insanely big
if (zipfile.getSize() > TOOBIG ) {
    throw new IllegalStateException("File to be unzipped is huge.");
}
```
Erroneous Best Practices (Spec. level)

- Only one code pattern specific for data compression
  - Rule: “IDS04-J. Safely extract files from ZipInputStream”

```java
// Write the files to the disk, but
// only if the file is not insanely big
if (zipfile.getSize() > TOOBIG)
    throw new IllegalStateException("File to be unzipped is huge.");
```

- `getSize()` returns ZIP file header with uncompressed size
Erroneous Best Practices (Spec. level)

- Only one code pattern specific for data compression
  - Rule: “IDS04-J. Safely extract files from ZipInputStream”
    
    ```java
    // Write the files to the disk, but
    // only if the file is not insanely big
    if (zipfile.getSize() > TOOBIG) {
        throw new IllegalStateException("File to be unzipped is huge.");
    }
    ```

- `getSize()` returns ZIP file header with uncompressed size

- but ZIP headers not integrity protected!
  - DoS countermeasure bypass

Notif. Authors
Conclusion
Conclusion/Takeaway

- Compression bombs are back
  - New vulnerabilities in popular network services

- ~20 years after the zip bombs, developers still unaware of the risks of handling data compression
  - 12 pitfalls which can be used by developers to build more secure services