Cross-App Security VI

Side Channels

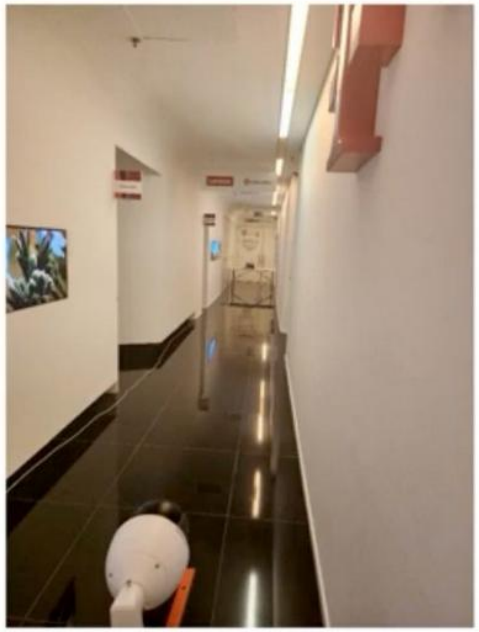
Side-Channel Attack

An attack that leverages information leaked as a byproduct of a system's operation

Hackers can steal cryptographic keys by video-recording power LEDs 60 feet away

Key-leaking side channels are a fact of life. Now they can be done by video-recording power LEDs.

DAN GOODIN – JUN 13, 2023 9:30 A.M. | 142



https://arstechnica.com/information-technology/2023/06/hackers-can-steal-cryptographic-keys-by-video-recording-connected-power-leds-60-feet-away/

"Researchers have devised a novel attack that recovers the secret encryption keys stored in smart cards and smartphones by using cameras in iPhones or commercial surveillance systems to video record power LEDs that show when the card reader or smartphone is turned on."

Timing-Based Side-Channel Attacks

- Rely on differences in timing that permit inferences of system state
- Timing differences can result from computational paths, network latency, memory access patterns, and more…
- Modern browsers implement some protections against timing attacks
- Mitigations exist against *Spectre*-like attacks as well as some other types of information leaks
- Nevertheless, numerous attack vectors still exist

Timing-Based Side-Channel Attacks: Example

- PIN-based lock that checks numbers in a sequence
- The system checks digit-by-digit
- As soon as an incorrect digit is encountered, the algorithm responds
- Each comparison takes 1 second

Timing-Based Side-Channel Attacks: Example

Actual PIN: 8473

Guess: 1111

- Rejected after ~ 1 second

Guess: 2111

- Rejected after ~1 second

Guess: 8111

- Rejected after ~2 second

```
function insecureVerify(inputPin) {
    if (inputPin.length !== SECRET_PIN.length) {
        return false;
    }

    for (let i = 0; i < SECRET_PIN.length; i++) {
        if (inputPin[i] !== SECRET_PIN[i]) {
            return false;
        }
    }

    return true;
}
```

Timing-Based Side-Channel Attacks: Example

- Digits can be guessed sequentially using time as a side-channel for correctness
- Without this, the maximum number of guesses would be 10^4 (10000)
- Instead, maximum number of guesses = 4 x 10 (40)
- In practice, noise can impact the number of required guesses
- Strategies:
  - Reduce noise
  - Conduct statistical analysis

Timing-Based Side-Channel Attacks: Browser

```
async function leakSensitiveData(keyword) {
    const url = `https://health-portal.com/search?q=${keyword}`;
    const start = performance.now();

    try {
        await fetch(url, { mode: 'no-cors', cache: 'no-store' });
    } catch (e) {
    }

    const duration = performance.now() - start;
    console.log(`Search for "${keyword}" took ${duration.toFixed(2)}ms`);

    // Interpretation:
    // If duration > 200ms, the user likely HAS results for this keyword.
    // If duration < 50ms, the user likely has NO results.
}

// The attacker tries a specific keyword
leakSensitiveData('Diabetes');
```

Timing-Based Side-Channel Attacks: Browser

Mitigations:

- **Eliminate the source of the side channel**
  - **Ensure responses take the same time to generate regardless of content**
- **Limit capability to send requests cross-app**
  - **Validate Fetch Metadata**
  - **Use Strict *SameSite* cookies**
  - **Implement strict *Cross-Origin-Resource-Policy***
  - **Validate workflow (like CSRF synchronizer pattern)**
- **Browsers implement default protections to reduce timing resolution**
  - ***performance.now()* is clamped by browsers with added jitter**
    - **Relaxed when Site Isolation achieved**

Compression-Based Side-Channel Attacks

- Lossless compression works by reducing redundancy
- Example: Substitution Code (Dictionary Method):
  - Starting phrase: "Rain, rain, go away, come again another day.“
  - Step 1: Identify repetition
    - “rain” appears 2 times
    - “ay” appears 2 times
  - Step 2: Create dictionary
    - 1 = rain
    - 2 = ay
  - Step 3: Write compressed version
    - "1, 1, go aw2, come again another d2.“
  - Note: this example is not effective compression because the result must include the dictionary

BREACH (2013)

BREACH = Browser Reconnaissance & Exfiltration via Adaptive Compression of Hypertext

Requirements:

1. **Attacker must be able to inject chosen plaintext into victim's requests**
2. **Attacker must be able to measure the size of encrypted traffic**
3. **Sensitive information is compressed**

Technical Approach:

1. **Attacker causes victim to navigate to attacker-controlled site and can initiate malicious Cross-Origin communicate to target app**
2. **Attacker sites on the same network as the victim and can intercept encrypted (TLS) communication**
3. **HTTP compression supported by target app**

Aside: HTTP Compression

- HTTP natively supports compression (commonly Gzip or Brotli)
- The browser requests compression and the server conforms
- Note: Burp automatically decompresses for you! (Repeater settings)

HTTP Request

```
GET /index.html HTTP/1.1
Host: example.com
Accept-Encoding: gzip, br
```

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Encoding: br

[Binary Compressed Data...]
```

BREACH: Example

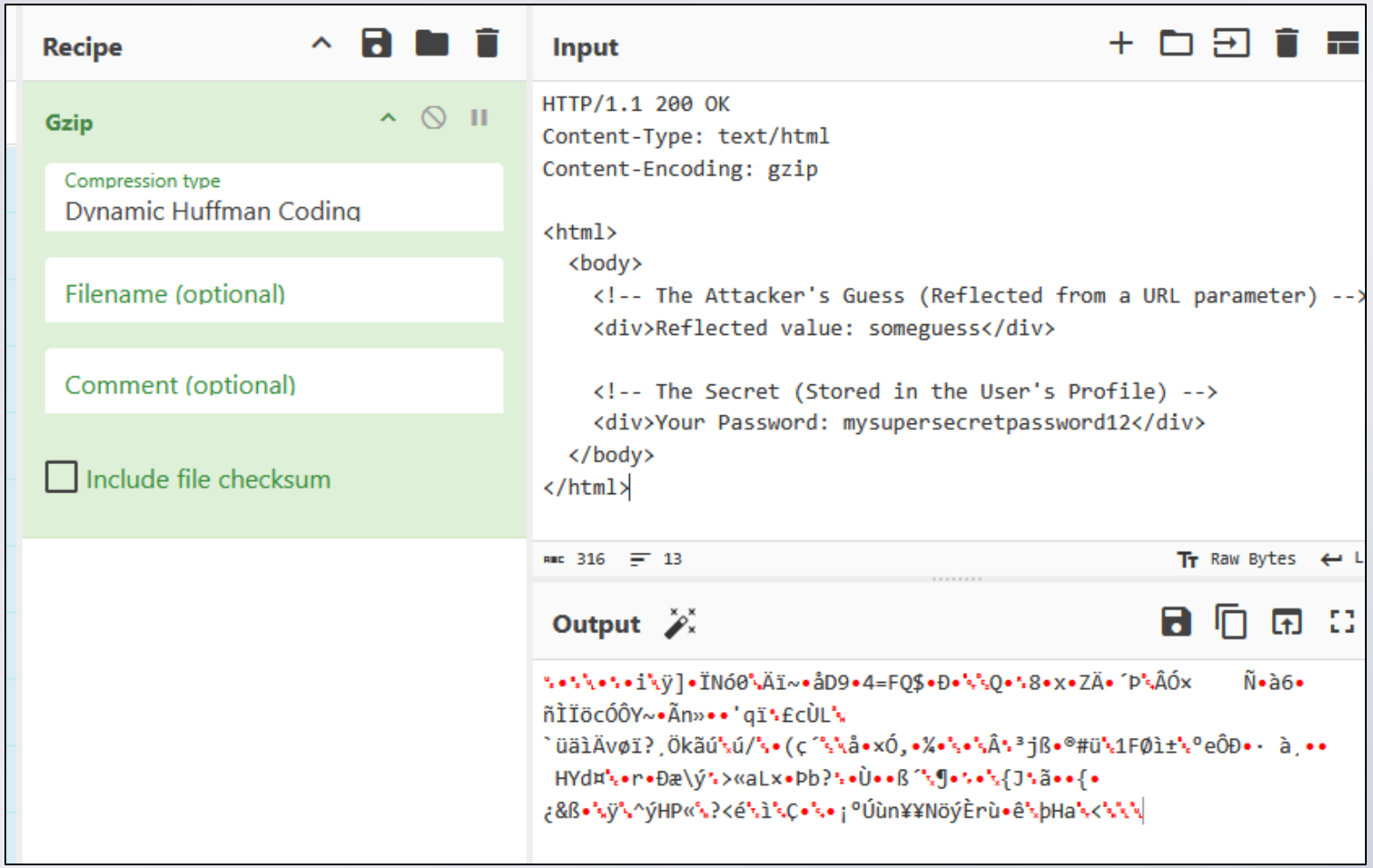
Consider a target web application that behaves as follows:

- **HTTP compression is used (applies to HTTP bodies)**
- **An endpoint exists to retrieve sensitive user data:**
  - **www.target-site.com/getData**
- **The response includes the user's password**
- **The same endpoint accepts and reflects a parameter value:**
  - **www.target-site.com/getData?reflectMe=someValue**
- **The app uses HTTP cookies with *SameSite=None* and implements no other cross-app restriction mechanisms**

DBG App Test Finding
BREACH Attack Potential

BREACH: Example (Cyber Chef)

Request: www.target-site.com/getData?reflectMe=someguess
Response Size (Compressed): 236



BREACH: Example

Password:
mysupersecretpassword12

Request: www.target-site.com/getData?reflectMe=someguess
Response Size (Compressed): 236

Request: www.target-site.com/getData?reflectMe=1234567890
Response Size (Compressed): 239

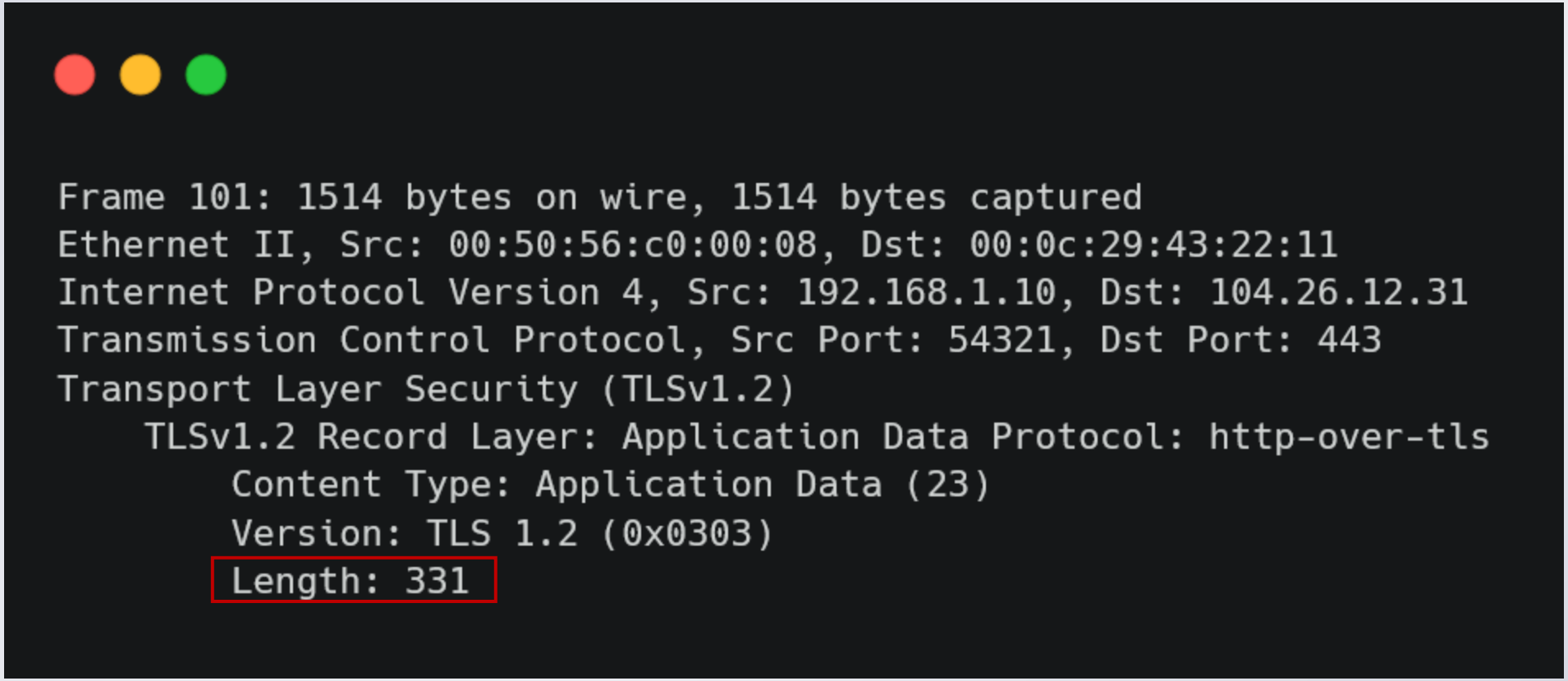
Request: www.target-site.com/getData?reflectMe=my
Response Size (Compressed): 230

Request: www.target-site.com/getData?reflectMe=myguess
Response Size (Compressed): 233

Request: www.target-site.com/getData?reflectMe=mysuper
Response Size (Compressed): 230

Request: www.target-site.com/getData?reflectMe=mysupersecret
Response Size (Compressed): 231

BREACH: Example (Wireshark)



BREACH: Mitigation

- Disable HTTP Compression
- Introduce random padding to obscure the length
- Implement cross-app protections
- Rearchitect the application to remove potentially vulnerable functionality
- Mask or remove secrets
- Enforce TLS v1.3
  - Adds packet padding that can obscure traffic analysis
- Enforce HTTP/2
  - Multiplexing may further obscure traffic

CRIME (2012)

- Compression Ratio Info-Leak Made Easy
- Basically identical to BREACH, but relied on TLS compression
- Unlike HTTP compression TLS included HTTP headers
- So BREACH, but for HTTP response headers (like Cookies)
  - However, CRIME did not depend as much on app behavior
- TLS compress-then-encrypt no longer utilized

Side-Channel Trade-Offs

- As seen, there is often a conflict between performance and security that arises with side channel attacks
- Recall the previous example..
- A secure alternative would complete every comparison rather than aborting early (constant time)
- “Timing-safe implementation”

```
function insecureVerify(inputPin) {
    if (inputPin.length !== SECRET_PIN.length) {
        return false;
    }

    for (let i = 0; i < SECRET_PIN.length; i++) {
        if (inputPin[i] !== SECRET_PIN[i]) {
            return false;
        }
    }

    return true;
}
```

PDF Compression Side-Channel Oracle

- Novel finding not reported previously
- PDF generators often use lossless text compression
- Therefore, applications that dynamically build PDFs may be vulnerable to BREACH-like exploitation without HTTP compression
- Realistically, not likely to be exploited…

Identifying Side-Channel Vulnerabilities

- Requires knowledge of present browser mechanisms/behavior
- Requires application functionality that can be abused in combination with browser capabilities
- Additional specific attacks will be covered in the XS-Leaks session

OWASP ASVS 5.0

Same requirements previously shared...

https://github.com/OWASP/ASVS/blob/master/5.0/en/