BakingTimer: Privacy Analysis of Server-Side Request Processing Time

Iskander Sanchez-Rola, Davide Balzarotti, and Igor Santos
Motivation

Cookies were originally introduced as a way to provide state awareness to websites, but nowadays they are not limited to store the login information or the current state of user. In several cases, third-party cookies are deliberately used for web tracking.
Motivation

Cookies were originally introduced as a way to provide state awareness to websites, but nowadays they are not limited to store the login information or the current state of user. In several cases, third-party cookies are deliberately used for web tracking.

But even if the most famous, cookies are not the only technique capable of retrieving the users’ browsing history. In fact, history sniffing techniques can do it without relying on any specific code in a third-party website, but only on code executed in one site.
Threat Model

Alice
Threat Model

Alice

Shopping
Threat Model

Alice

Shopping

News
Threat Model

Alice

Shopping

News
Threat Model

Alice

Shopping

News
Alice

Threat Model

Shopping

News
Threat Model

Alice

Shopping

News
Threat Model

Alice

Shopping

News
Threat Model

Alice

Shopping

News
Threat Model

Alice

Shopping

News
Threat Model

Alice

Shopping

News
Threat Model

Alice

Shopping

News
Threat Model

Alice

Shopping

News
Threat Model

Alice

Streaming

Shopping

News
Threat Model

Alice

Streaming

Shopping

News
Threat Model

- Alice
- Streaming
- Shopping
- News
Threat Model

Alice

Streaming  Shopping  News
Threat Model
Threat Model

Alice

Streaming

Shopping

News
BakingTimer

We present a new timing side-channel attack, that relies on the presence of first party cookies set by the target websites to perform history sniffing.
BakingTimer

We present a new timing side-channel attack, that relies on the presence of first party cookies set by the target websites to perform history sniffing.

Our system is based on the analysis of how servers process HTTP requests, and by using this information, is able to detect both if the user previously visited the website and whether she is currently logged in.
BakingTimer

The main observation behind our approach is that, when the browser sends a cookie along with a request, it is reasonable to assume that the server will check its value.
BakingTimer

The main observation behind our approach is that, when the browser sends a cookie along with a request, it is reasonable to assume that the server will check its value.

Then, it may use the value to retrieve the associated user session and load additional data from the database, or that it will simply execute a different path with respect to a request that does not contain any cookie.
BakingTimer

```php
<?php
$userID = "0bc63ecec05112d03544fde0b5a18c70";

?>
```
BakingTimer

```php
<?php
$userID = "0bc63ecec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"])) {

<?>

```
BakingTimer

```php
<?php
$userID = "0bc63ec0e05112d03544fde0b5a18c70";
if (isset($_COOKIE["consent"])) {
    //...}
?>
```

Copyright © 2019 NortonLifeLock Inc. All rights reserved.
BakingTimer

```php
<?php
$userID = "0bc63ec05112d03544fde0b5a18c70";
if (isset($_COOKIE["consent"])) {
    //
} else {
    askConsent();
}
?>
```
we assume an attacker can run JavaScript code on the client browser versions of Google Chrome [19].

In the timing attack presented in this paper, we adopt the same threat model used by previous work in the area [5].

In fact, all existing techniques fall in the classic "arms race" category, where each new defense is met by a new attack. Despite these mitigations, recent work has shown that even across a network connection, to tell the three behaviors apart is currently logged in into certain websites by timing of specific data that can result in serious security issues, such as the ones proposed by Smith et al. [7].

However, several possible defenses exist to avoid the problem, such as the ones proposed by Smith et al. [7].

For instance, Figure 1 shows a simple PHP skeleton that emulates a toy application, emphasizing the differences among the three possible cases. First, the program checks if the cookie is present at all, and if it is, it validates the user's state:

```
<?php
$userID = "0bc63ec05112d0354fde0b5a18c70";

if (isset($_COOKIE["consent"])) {
    // user is already logged in
    // perform additional checks
    // case A
}
else {
    askConsent(); // case B
}
?>
```

CASE A

Copyright © 2019 NortonLifeLock Inc. All rights reserved.
BakingTimer

```
<?php
$userID = "0bc63ece05112d03f4de0b5a18c70";

if (isset($_COOKIE["consent"])) {

} else {
    askConsent();
}
```
BakingTimer

```php
<?php
$userID = "0bc63ec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"])) {
    // code here
}
else {
    askConsent();
}
?>
```

Figure 1: Example code of a PHP server presenting the three possible cases of a cookie process schema.
BakingTimer

```php
<?php
$userID = "0bc63ec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"])) {
    if (isset($_COOKIE["userID"]) {

    } else {
        askConsent();
    }
} ?>
```

---

**CASE A**

In the timing attack presented in this paper, we adopt the same threat model used by previous work in the area [19]. In particular, for the CSS resource size of certain resources [18], In fact, [17]. The information collected by our technique allows an attacker to perform cross-origin requests. This code can be either loaded directly by the browser functions (e.g., `querySelector`), or by a third-party service (e.g., an advertisement or analytics company). Moreover, an interested tracker could create a list of websites and generate a temporal fingerprint against particular victims.

Out of all the requests a web server receives, some contain cookies against other web tracking techniques. Finally, from a security point of view, this information can be used to perform targeted attacks and therefore to accurately identify whether or not cookies are used.

Some of the attacks discussed above are already mitigated by browser vendors. For instance, for the CSS resource size of certain resources [18], In fact, [17]. The information collected by our technique allows an attacker to perform cross-origin requests. This code can be either loaded directly by the browser functions (e.g., `querySelector`), or by a third-party service (e.g., an advertisement or analytics company). Moreover, an interested tracker could create a list of websites and generate a temporal fingerprint against particular victims.

Out of all the requests a web server receives, some contain cookies against other web tracking techniques. Finally, from a security point of view, this information can be used to perform targeted attacks and therefore to accurately identify whether or not cookies are used.

Some of the attacks discussed above are already mitigated by browser vendors.
BakingTimer

```php
<?php
    $userID = "0bc63ec05112d03544fde0b5a18c70";
    if (isset($_COOKIE["consent"])) {
        if (isset($_COOKIE["userID"])) {
            //... (remaining code)

    } else {
        askConsent();
    }

 CASE A
```
BakingTimer

```php
<?php

$userId = "0bc63ec05112d354fde05a18c70";

if (isset($_COOKIE["consent"])) {
    if (isset($_COOKIE["userId"])) {

    } else {
        saveNavigation();
    }

} else {
    askConsent();
}
?>
```

Figure 1: Example code of a PHP server presenting the three possible cases of a cookie process schema.

Figure 2: Server cookie process schema.
BakingTimer

```php
<?php
$userID = "0bc63ec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"])) {
    if (isset($_COOKIE["userID"])) {

    } else {
        saveNavigation();
    }

} else {
    askConsent();
}
?>
```
BakingTimer

```php
<?php
$userId = "0bc63ecec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"])) {
    if (isset($_COOKIE["userId"])) {
        // ...
    } else {
        saveNavigation();
    }
} else {
    askConsent();
}
?>
```
BakingTimer

```php
<?php
$ userID = "bc63ecec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"])) {
    if (isset($_COOKIE["userID"])) {
        // CASE A
        saveNavigation();
    } else {
        // CASE B
        getUserData();
    }
} else {
    // CASE C
    askConsent();
}
?>
```
BakingTimer

```php
<?php
$userID = "0bc63ec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"])) {
    if (isset($_COOKIE["userID"])) {

    } else {
        saveNavigation();
    }
} else {
    askConsent();
}
?>
```
we assume an attacker can run JavaScript code on the client browser versions of Google Chrome [19].

The information collected by our technique allows an attacker to determine which websites were previously visited by the user directly by the browser functions (e.g., getUserData, getComputedStyle). Despite these mitigations, recent work has shown that one of these new techniques has already been blocked in recent browsers. However, several possible defenses exist to avoid the attack is still possible using new features available in modern browsers. Therefore, we decided to

Out of all the requests a web server receives, some contains cookies submitted alongside an HTTP request. In particular, there are two main tests that make detecting a timing difference possible: the first to verify if there are cookies at all and the second to analyze them. While the comparison themselves are too large to consider in this work, we rely on two simple cases: the first is the presence of the cookies, the second is the absence of them.

Some of the attacks discussed above are already mitigated by browser vendors. For instance, for the CSS fingerprinting capabilities, the user agent string is used to identify the browser type. Although this parameter is not used to identify the browser version, it can be used to infer the browser model and the version. In fact, all existing techniques fall in the classic "arms race" category, in which attacker and researchers constantly discover new tricks that are in turn mitigated by browser vendors, website developers, or even simply careful user settings. Therefore, we decided to

Countermeasures and Shortcomings

In fact, all existing techniques fall in the classic "arms race" category, in which attacker and researchers constantly discover new tricks that are in turn mitigated by browser vendors, website developers, or even simply careful user settings. Therefore, we decided to

Figure 1: Example code of a PHP server presenting the three possible cases of a cookie process schema.

```php
<?php
$userID = "0bc63ecec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"])) {
    if (isset($_COOKIE["userID"])) {
        //
    } else {
        saveNavigation();
    }
} else {
    askConsent();
}
?>
```

Figure 2: Server cookie process schema.
we assume an attacker can run JavaScript code on the client browser multiple usages for this data that can result on serious security and on which website the user is currently logged in. There are to determine which websites were previously visited by the user directly by the to perform cross-origin requests. This code can be either loaded threat model used by previous work in the area [2].

2.3 Threat Model

In the timing attack presented in this paper, we adopt the same category, in which attacker and researchers constantly discover new

problem, such as the ones proposed by Smith et al. [17].

However, several possible defenses exist to avoid the attack is still possible using new features available in modern browsers. However, several possible defenses exist to avoid the

example code of a PHP server presenting the three

requests: [Figure 1:]

BakingTimer: Privacy Analysis of Server-Side Request Processing Time ACSAC '19, December 9–13, 2019, San Juan, PR, USA

The information collected by our technique allows an attacker in fact, all existing techniques fall in the classic "arms race" cate-

Some of the attacks dis-

Figure 2: Server cookie process schema.

Out of all the requests a web server receives, some contain cookies and some do not. The main observation behind our approach is that, when the browser sends a cookie along with an HTTP request, it is able to always return that the user has never visited the link [30]. Despite these mitigations, recent work has shown that even across a network connection, to tell the three behaviors apart, all the corresponding

of various users, indicating the user's state in each of them. Even if theنصر

Copyright © 2019 NortonLifeLock Inc. All rights reserved. 42
BakingTimer

```php
<?php

$userID = "0bc63ec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"])) {
    if (isset($_COOKIE["userID"])) {
        if ($_POST["userID"] == $userID) {
            getUserData();
        }
    } else {
        saveNavigation();
    }
} else {
    askConsent();
}
?>
```
In the timing attack presented in this paper, we adopt the same threat model used by previous work in the area [19].

The information collected by our technique allows an attacker to determine which websites were previously visited by the user directly by the server-side application. This code can be either loaded by an advertisement or analytics company (e.g., for the CSS resource size of certain resources [15]), or by estimating the execution time of the footprint for the client browser versions of Google Chrome [19].

The information stored in the cookies. If "userID" is not indicated, the program performs additional checks by looking for some login consent or because it has been deleted), the program takes the path that the server-side application will check its presence among the three cases is sufficient, and the corresponding additional data from the database, or that it will simply execute the path that makes detecting a timing difference among the three behaviors apart easier.

In fact, all existing techniques fall in the classic "arms race" category, in which attacker and researchers constantly discover new tricks that are in turn mitigated by browser vendors, website developers (e.g., for the CSS resource size of certain resources [15]), or by estimating the execution time of the footprint for the client browser versions of Google Chrome [19].

Some of the attacks discussed above are already mitigated by browser vendors. For instance, Figure 1 shows a simple PHP skeleton that emulates three different possible cases of a cookie process schema.

```php
<?php
$userID = "0bc63ecec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"]) {
    if (isset($_COOKIE["userID"]) {
        if ($_POST["userID"] == $userID) {
            getUserData();
            return;
        }
    } else {
        saveNavigation();
    }
} else {
    askConsent();
}?>
```

Figure 1: Example code of a PHP server presenting the three different possible cases of a cookie process schema.
BakingTimer

```php
<?php
$userID = "0bc63ecec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"])) {
    if (isset($_COOKIE["userID"])) {
        if ($_POST["userID"] == $userID) {
            getUserData();
        } else {
            saveNavigation();
        }
    } else {
        askConsent();
    }
}?>
```

Figure 2 shows a simplified representation of the control-flow possible: the three behaviors among the three cases is sufficient, our hypothesis.

Case A
- else { askConsent(); }

Case B
- else {
  saveNavigation();
}

Case C
- getUserData();
we assume an attacker can run JavaScript code on the client browser versions of Google Chrome [19].

In fact, all existing techniques fall in the classic "arms race" category, in which attacker and researchers constantly discover new problems, such as the ones proposed by Smith et al. [1]. Despite these mitigations, recent work has shown that the attack is still possible using new features available in modern browsers. However, several possible defenses exist to avoid the problem, such as the ones proposed by Smith et al. [1].

The threat model used by previous work in the area [2,3,4] investigates if it was possible to devise a new technique that would 1) be more general than the previous attacks, 2) be more difficult to mitigate, and 3) be useful for a wide range of attackers.

Out of all the requests a web server receives, some contain cookies. Therefore, the server-side application will check its cookies when the browser sends a cookie along with an HTTP request, it is reasonable to assume that the server-side application will check its cookies and therefore to accurately identify whether or not cookies are present.

There are multiple usages for this data that can result in serious security and privacy implications. The most obvious is related to advertisement, as the usage of the browsing history allows to display targeted advertisements. Moreover, an interested tracker could create a temporal list of websites and generate a timestamp fingerprint to verify if there are cookies at all and the second to analyze them.
BakingTimer

```php
<?php
$userID = "0bc63ec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"]) { 
    if (isset($_COOKIE["userID"])) {
        if ($_POST["userID"] == $userID) {
            getUserData();
        }
        else {
            saveNavigation();
        }
    } else {
        askConsent();
    }
} else {
    askConsent();
}?>
```
BakingTimer

```php
<?php
$UserID = "0bc63ec05112d0354fde0b5a18c70";

if (isset($_COOKIE["consent"])) {
    if (isset($_COOKIE["UserID"])) {
        if ($_POST["UserID"] == $UserID) {
            getUserData();
        }
    }
} else {
    saveSession();
}
else {
    askConsent();
}
?>
```
BakingTimer

```php
<?php
$userID = "0bc63ecec05112d03544fde0b5a18c70";

if (isset($_COOKIE["consent"])) {
    if (isset($_COOKIE["userID"])) {
        if ($_POST["userID"] == $userID) {
            getUserData();
        }
    }
    else {
        saveNavigation();
    }
} else {
    askConsent();
}
?>
```
The information collected by our technique allows an attacker to infer sensitive data such as the user’s state on the website or because it has been deleted, the program takes the path when the browser sends a cookie along with an HTTP request, it is submitted alongside an HTTP request. In particular, there are two main tests that make detecting a timing difference possible: the first tests for the presence of cookies and therefore to accurately identify whether or not cookies are present, and the second to analyze them.

Out of all the requests a web server receives, some contain cookies information is collected by our technique. An attacker can analyze the server-side request processing time of various users, indicating the user’s state in each of them. Even if the attacker cannot obtain the user’s session data, the server-side application can perform additional checks by looking for some login information. When the user already accessed the website before, the server-side application executes the routines necessary to create new cookies. If a cookie is not found (either because it is the first time the user access the website or because it has been deleted), the program follows a different execution path with respect to a request that does not contain any cookie (in which case, for example, the application may contain additional data from the database, or that it will simply execute the routines necessary to create new cookies).

In practice, for the CSS tricks that are in turn mitigated by browser vendors, website developers need to rely only on server-side information, and 2) that could not be easily prevented without degrading the performance or functionalities of the web application.

Some of the attacks discussed above are already mitigated by browser vendors. For instance, Figure 1 shows a simple PHP skeleton that em-
REQUEST 1 + REQUEST 2
BakingTimer

REQUEST 1
xmlHttprequest.withCredentials = FALSE;
we assume an attacker can run JavaScript code on the client browser multiple usages for this data that can result in serious security to determine which websites were previously visited by the user by an advertisement or analytics company). direct by the threat model used by previous work in the area [2.3 Threat Model a web application. prevented without degrading the performance or functionalities of ops, or even simply careful user settings. Therefore, we decided to tricks that are in turn mitigated by browser vendors, website develop-
gory, in which attacker and researchers constantly discover new one of these new techniques has already been blocked in recent problem, such as the ones proposed by Smith et al. [have been modi-browser functions (e.g., for the CSS discussed above are already mitigated by browser vendors. For in-
Countermeasures and Shortcomings

The information collected by our technique allows an attacker different possible cases of a cookie process schema.

Figure 1: Example code of a PHP server presenting the three

Figure 2 shows a simpli-

For instance, Figure 1 shows a simple PHP skeleton that em-

Out of all the requests a web server receives, some contains cookies

3 BAKINGTIMER against particular victims.

of various users, indicating the user's state in each of them. Even if

tate a prede-

erent execution path with respect to a request that does not

erent paths. Our hypothe-

BakingTimer

Request → Cookies?

S
E
R
V
E
R

Case A

Case B

Case C

Copyright © 2019 NortonLifeLock Inc. All rights reserved.

BakingTimer

Copyright © 2019 NortonLifeLock Inc. All rights reserved.
BakingTimer

![Diagram of BakingTimer](image)

The information collected by our technique allows an attacker to perform targeted attacks on a webpage or because it has been deleted), the program takes the path to verify if there are cookies at all and the second to analyze them one by one. As explained in Section 3, our own comparison themselves are too fine to be used, so we named our technique BakingTimer. For instance, Figure 1 shows a simple PHP skeleton that embeds a cookie process schema. Out of all the requests a web server receives, some contains cookies and some do not. The main observation behind our approach is that, in case of a cookie process schema, the program performs additional checks by looking for some login or request. For instance, if the cookie is not found (either because it is the first time the user has accessed the website before, by testing for the presence of the user ID in the cookie), the program may execute the routines necessary to create new cookies. However, if the cookie contains any cookie (in which case, for example, the application may contain additional data from the database, or that it will simply execute one of these new techniques has already been blocked in recent work, such as the ones proposed by Smith et al. [20].)

Countermeasures and Shortcomings

A technique to prevent the attack belongs to the category, in which attacker and researchers constantly discover new ways to perform some action. Despite these mitigations, recent work has shown that the attack is still possible using new features available in modern web browsers. In particular, Figure 2 shows a simplified representation of the control-flow graph that results from a request to a server, in which attacker and researchers constantly discover new ways to perform some action. In fact, as the usage of the browsing history allows to display different possible cases of a cookie process schema, for instance, the usage of the browsing history allows to display different possible cases of a cookie process schema. The main tests that make detecting a timing difference possible: the time difference among the three cases is sufficient to infer whether there is a cookie or not. In particular, the time difference among the three cases is sufficient to infer whether there is a cookie or not.

Some of the attacks discussed above are already mitigated by browser vendors. For instance, for the CSS property and load session data. While the comparison themselves are too fine to be used, so we named our technique BakingTimer. For instance, Figure 1 shows a simple PHP skeleton that embeds a cookie process schema. Out of all the requests a web server receives, some contains cookies and some do not. The main observation behind our approach is that, in case of a cookie process schema, the program performs additional checks by looking for some login or request. For instance, if the cookie is not found (either because it is the first time the user has accessed the website before, by testing for the presence of the user ID in the cookie), the program may execute the routines necessary to create new cookies. However, if the cookie contains any cookie (in which case, for example, the application may contain additional data from the database, or that it will simply execute one of these new techniques has already been blocked in recent work, such as the ones proposed by Smith et al. [20].)

Countermeasures and Shortcomings

A technique to prevent the attack belongs to the category, in which attacker and researchers constantly discover new ways to perform some action. Despite these mitigations, recent work has shown that the attack is still possible using new features available in modern web browsers. In particular, Figure 2 shows a simplified representation of the control-flow graph that results from a request to a server, in which attacker and researchers constantly discover new ways to perform some action. In fact, as the usage of the browsing history allows to display different possible cases of a cookie process schema, for instance, the usage of the browsing history allows to display different possible cases of a cookie process schema. The main tests that make detecting a timing difference possible: the time difference among the three cases is sufficient to infer whether there is a cookie or not. In particular, the time difference among the three cases is sufficient to infer whether there is a cookie or not.

Some of the attacks discussed above are already mitigated by browser vendors. For instance, for the CSS property and load session data. While the comparison themselves are too fine to be used, so we named our technique BakingTimer. For instance, Figure 1 shows a simple PHP skeleton that embeds a cookie process schema. Out of all the requests a web server receives, some contains cookies and some do not. The main observation behind our approach is that, in case of a cookie process schema, the program performs additional checks by looking for some login or request. For instance, if the cookie is not found (either because it is the first time the user has accessed the website before, by testing for the presence of the user ID in the cookie), the program may execute the routines necessary to create new cookies. However, if the cookie contains any cookie (in which case, for example, the application may contain additional data from the database, or that it will simply execute one of these new techniques has already been blocked in recent work, such as the ones proposed by Smith et al. [20].)
BakingTimer

REQUEST 2
BakingTimer

REQUEST 2

```javascript
xmlHttpRequest.withCredentials = TRUE;
```
BakingTimer

Request ➔ Cookies?

S
E
R
V
E
R
BakingTimer

Request → Cookies?  

Case A

\[ x \text{ time} \]
we assume an attacker can run JavaScript code on the client browser, multiple usages for this data that can result in serious security implications. To determine which websites were previously visited by the user, an advertisement or analytics company may directly perform cross-origin requests. This code can be either loaded directly or loaded by an advertisement or analytics company).

directly by the threat model used by previous work in the area.

2.3 Threat Model

A web application can be attacked by exploiting the presence of cookies. This attack can be divided into three cases:

1. The browser sends a cookie along with an HTTP request.
2. The server-side application checks its cookies.
3. The server-side application does not check its cookies.

The main tests that make detecting a timing difference possible: the difference in the time it takes to execute the routines necessary to create new cookies.

Out of all the requests a web server receives, some contain cookies, the presence of other web tracking techniques. Finally, from a security point of view, this information can be used to perform targeted attacks against particular victims.

In fact, the information stored in the cookies. If userID is not indicated, the program performs additional checks by looking for some login information.

Some of the attacks discussed above are already mitigated by browser vendors. For instance, the usage of the browsing history allows to display different possible cases of a cookie process schema.

The information collected by our technique allows an attacker to verify if there are cookies at all and the second to analyze them.

Countermeasures and Shortcomings

The link to the control file is currently logged in into certain websites by timing of specific requests.

The information collected by our technique allows an attacker to determine which websites were previously visited by the user.

Case A

Figure 2 shows a simplified representation of the control file. The browser sends a cookie along with an HTTP request, it is assumed that the server-side application will check its cookies. If userID is not indicated, the program performs additional checks by looking for some login information.
we assume an attacker can run JavaScript code on the client browser versions of Google Chrome [19].

In the timing attack presented in this paper, we adopt the same threat model used by previous work in the area [30].

The information collected by our technique allows an attacker to determine which websites were previously visited by the user directly by the browser functions (e.g., getUserData()).

The link [35] has been modiﬁed to always return that the user has never visited a web application.

There are several possible defenses exist to avoid the problem, such as the ones proposed by Smith et al. [44]. In particular, for the CSS selectors, all the corresponding tricks that are in turn mitigated by browser vendors, website developers.

In fact, all existing techniques fall in the classic “arms race” category, in which attacker and researchers constantly discover new tricks that are in turn mitigated by browser vendors, website developers, or by estimating the size of certain resources [30].

In the timing attack presented in this paper, we adopt the same threat model used by previous work in the area [30].

Out of all the requests a web server receives, some contains cookies submitted alongside an HTTP request. In particular, there are two main tests that make detecting a timing difference possible: the first validated, and then the application follows a different execution path with respect to a request that does not contain cookies.

The main observation behind our approach is that, from the server point of view, this information can be used to perform targeted attacks against particular victims.

The information stored in the cookies. If userID is not indicated, it is not found (either because it is the expected case or that it will simply execute the routines necessary to create new cookies).

The main observation behind our approach is that, in particular, for the CSS selectors, all the corresponding tricks that are in turn mitigated by browser vendors, website developers, or by estimating the size of certain resources [30].

The main observation behind our approach is that, in particular, for the CSS selectors, all the corresponding tricks that are in turn mitigated by browser vendors, website developers.

The main observation behind our approach is that, in particular, for the CSS selectors, all the corresponding tricks that are in turn mitigated by browser vendors, website developers.
we assume an attacker can run JavaScript code on the client browser versions of Google Chrome [19].

multiple usages for this data that can result on serious security and on which website the user is currently logged in. There are to determine which websites were previously visited by the user by an advertisement or analytics company).

directly by the

to perform cross-origin requests. This code can be either loaded

threat model used by previous work in the area [2.3 Threat Model

a web application.

rely only on server-side information, and 2) that could not be easily

investigate if it was possible to devise a new technique that would 1)

opers, or even simply careful user settings. Therefore, we decided to

tricks that are in turn mitigated by browser vendors, website devel-
gory, in which attacker and researchers constantly discover new

one of these new techniques has already been blocked in recent

browsers. However, several possible defenses exist to avoid the

the link [2.3 Threat Model

browser functions (e.g.,

stance, for the CSS

Countermeasures and Shortcomings

size of certain resources [2.3 Threat Model

requests [2.3 Threat Model

is currently logged in into certain websites by timing of speci

di

Figure 1: Example code of a PHP server presenting the three

BakingTimer: Privacy Analysis of Server-Side Request Processing Time ACSAC '19, December 9–13, 2019, San Juan, PR, USA

Figure 2: Server cookie process schema.

Case A

Case B

Case C

Request -> Cookies? -> which? -> login

yes -> access -> Case C

no -> Case A

x time

y time

z time

x time

y time

z time

18

17

16

15

14

13

12

11

10

9

8

7

6

5

4

3

2

1


Experiment Dataset
Access Detection

PHASE 1 + PHASE 2
Access Detection

PHASE 1
Access Detection

PHASE 1

NEVER VISITED
Access Detection
Access Detection
Access Detection
Access Detection
Access Detection
Access Detection

PHASE 2
Access Detection

PHASE 2

PREVIOUSLY VISITED
Access Detection
Access Detection
Access Detection
Access Detection
Access Detection
Access Detection
Access Detection
Access Detection

- NEVER VISITED
- PREVIOUSLY VISITED
Access Detection

More than **half of the websites analyzed** were vulnerable to our attack. More concretely, around 70% with private personal information, and around 40% of highly accessible.
Access Detection

More than half of the websites analyzed were vulnerable to our attack. More concretely, around 70% with private personal information, and around 40% of highly accessible.

We compared the mean and standard deviation of the number of cookies, and results show that highly accessed websites have a higher number of cookies. This hints that slower servers or less optimized code seem the resposible of the difference.
Login Detection

We can also check if the user is logged in. In our dataset, we found highly accessible websites such as World of Warcraft (WoW) or Gucci, and websites related to private personal information such as LGBTchat or Dynamic Catholic.
Login Detection

We can also check if the user is logged in. In our dataset, we found highly accessible websites such as World of Warcraft (WoW) or Gucci, and websites related to private personal information such as LGBTchat or Dynamic Catholic.

Curiously, some websites do not properly delete all cookies related to the login, what we call persistent login. In this cases, it is possible to detect a previous logged-in state even if not logged at that moment (e.g., Microsoft/MSN and Openload).
Stability Test
Stability Test
Stability Test
Countermeasures

Regular defenses for server-side timing attacks include, a random delays in the response time, or fixed response times for sensitive requests. But are difficult and impractical to implement in reality due to performance issues.
Countermeasures

Regular defenses for server-side timing attacks include, a random delays in the response time, or fixed response times for sensitive requests. But are difficult and impractical to implement in reality due to performance issues.

Another option would be cookies with the SameSite attribute, that can indicate that they do not want to be send in third-party requests. However, as long as one of the cookies involved does not indicate it, the attack would still work.
Thank You!

iskander_sanchez@symantec.com
iskander-sanchez-rola.com