CORS. Аутентификация

Самосадный Кирилл
https://t.me/kirsamosad
samosad@seclab.cs.msu.su

МГУ 2018

Same Origin Policy

Как атаковать жертву?

- Cookie с флагами secure и httponly могут быть перезаписаны методом переполнения cookie jar
- Нельзя надежно разделить приложения различного уровня критичности, например:
  - payments.site.com от blogs.site.com
- Пусть на blogs.site.com есть XSS
- Пусть на payments.site.com cookie выставлены как secure, httponly и без domain

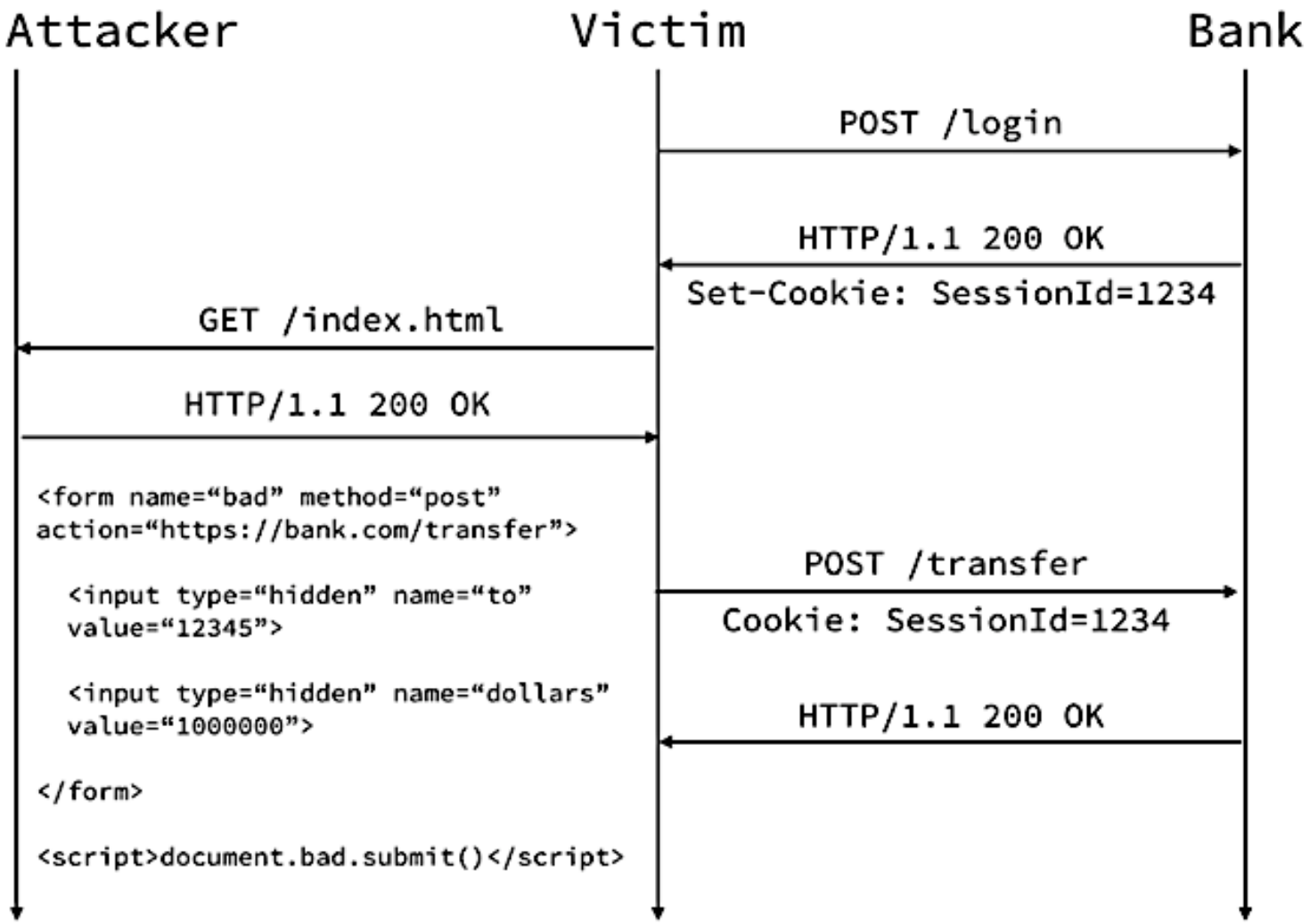
Ответ

- Как атаковать жертву?
  - выставляем через XSS cookie с domain = site.com
  - cookie jar для blogs.site.com переполняется и вытесняет cookies пользователя
  - выставляем свои cookies
  - порядок и кол-во посылаемых cookies зависит от браузера

DEMO: Session fixation

CSRF

Cross Site Request Forgery

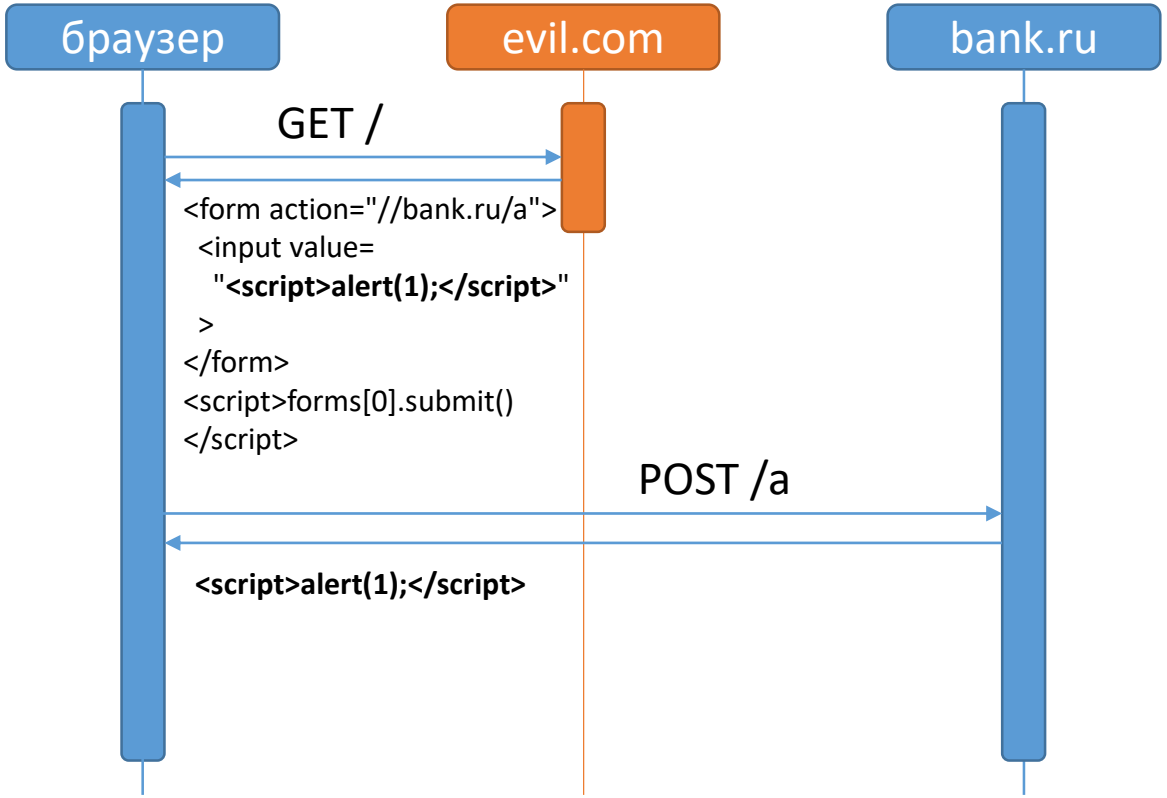


DEMO: CSRF

CSRF: обеспечить проверку аутентичности запросов

- Сделать формат запроса непредсказуемым
  - CAPTCHA
  - Token
  - ввод пароля/дополнительного фактора
- Аутентифицировать источник запроса
  - проверять Referer (плохо)
  - выставлять свои заголовки для XHR и проверять их наличие
- Сделать сессию на заголовках
- Не ошибиться при конфигурации CORS (если актуально)
- Проверять CSRF централизованно

XSS over CSRF



CORS

Cross Origin Resource Sharing

- Позволяет **отправлять** XHR-запрос к другому домену и **прочитать** ответ
- Прочитать – это про SOP и XSS
- Отправить – это про CSRF

Археология

- Какие запросы можно вынудить отправить браузер до появления XHR?
- Что поменялось с введением XHR?
- Разработка CORS и при чем тут CSRF?

Cross Origin Resource Sharing

- Есть понятие "простого" запроса и "preflighted"
- Простой:
  - GET, HEAD или POST с Content-Type application/x-www-form-urlencoded, multipart/form-data, text/plain
  - без дополнительных заголовков, кроме Accept, Accept-Language, Content-Language
- Простой можно отправить всегда, но аналог итак можно отправить средствами HTML

Preflighted-запрос в коде

```
var invocation = new XMLHttpRequest();
var url = 'http://bar.other/resources/post-here/';
var body = '<?xml version="1.0"?><person><name>Arun</name></person>';

function callOtherDomain(){
  if(invocation)
    {
      invocation.open('POST', url, true);
      invocation.setRequestHeader('X-PINGOTHER', 'pingpong');
      invocation.setRequestHeader('Content-Type', 'application/xml');
      invocation.onreadystatechange = handler;
      invocation.send(body);
    }

......
```

Preflight в сети

```
OPTIONS /resources/post-here/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1b3pre)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Origin: http://foo.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER
```

Ответ на Preflight в сети

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:15:39 GMT
Server: Apache/2.0.61 (Unix)
Access-Control-Allow-Origin: http://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER
Access-Control-Max-Age: 1728000
Vary: Accept-Encoding, Origin
Content-Encoding: gzip
Content-Length: 0
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: text/plain
```

DEMO: Preflighted request

CORS and credentials

```
1  var invocation = new XMLHttpRequest();
2  var url = 'http://bar.other/resources/credentialed-content/';
3
4  function callOtherDomain(){
5    if(invocation) {
6      invocation.open('GET', url, true);
7      invocation.withCredentials = true;
8      invocation.onreadystatechange = handler;
9      invocation.send();
10   }
```

- **Запрос простой, поэтому preflight не будет**
- **Браузер не отдаст скрипту ответ, если в нем не будет одного из следующих заголовков:**
  - **Access-Control-Allow-Credentials: true**
  - **Access-Control-Allow-Origin: http://bar.other**
- **Браузер не отдаст скрипту ответ, если в нем будет Access-Control-Allow-Origin: ***
  - **справедливо для данного примера (withCredentials = True)**

DEMO: CORS

CORS: типичные ошибки

- Упрощение жизни для разработчика
  - Отдать в ответ в Access-Control-Allow-Origin то, что пришло в запросе в Origin
  - Разрешение на доступ из публичных тестовых сервисов (например, JSFiddler или CodePen)
- Неправильное использование wildcard и регулярных выражений ('*'):
  - notdomain.com
  - domain.com.evil.com
- sub.domain.com (и XSS на любом поддомене)
- Разрешение для HTTP => Man-In-The-Middle
- Origin: null может получить iframe на любом сайте (см. следующий слайд)
- кэши

CORS: Origin: null

Запрос:

GET /reader?url=zxcvbn.pdf
Host: docs.google.com
Origin: null

Ответ:

HTTP/1.1 200 OK
Access-Control-Allow-Origin: null
Access-Control-Allow-Credentials: true

Эксплоит на любом сайте:

<iframe sandbox="allow-scripts allow-top-navigation allow-forms"
src='data:text/html,<script>*cors stuff here*</script>'></iframe>

Но этого еще не достаточно

- Возможности:
  - HTML: iframe
  - JavaScript: возможность динамически менять размеры и положения элементов, навешивать и снимать обработчики событий, менять курсор
  - CSS: свойство opacity, свойство cursor
- В общем случае атака называется UI Redressing и имеет много видов: clickjacking, strokejacking, cursorjacking, likejacking
- Защита: X-Frame-Options: deny/sameorigin

Content Security Policy

- Facebook.com

default-src * data: blob:;

script-src *.facebook.com *.fbcdn.net *.facebook.net *.google-analytics.com *.virtualearth.net *.google.com 127.0.0.1:* *.spotilocal.com:* 'unsafe-inline' 'unsafe-eval' *.atlassolutions.com blob: data: 'self';

style-src data: blob: 'unsafe-inline' *;

connect-src *.facebook.com facebook.com *.fbcdn.net *.facebook.net *.spotilocal.com:* wss://*.facebook.com:* https://fb.scanandcleanlocal.com:* *.atlassolutions.com attachment.fbsbx.com ws://localhost:* blob: *.cdninstagram.com 'self' chrome-extension://boadgeojelhgndaghljhdicfkmllpafd chrome-extension://dliochdbjfkdbacpmhlcpmleaejidimm;

- frame-ancestors none;

Аутентификация и авторизация в микросервисной архитектуре

Есть проблемы?

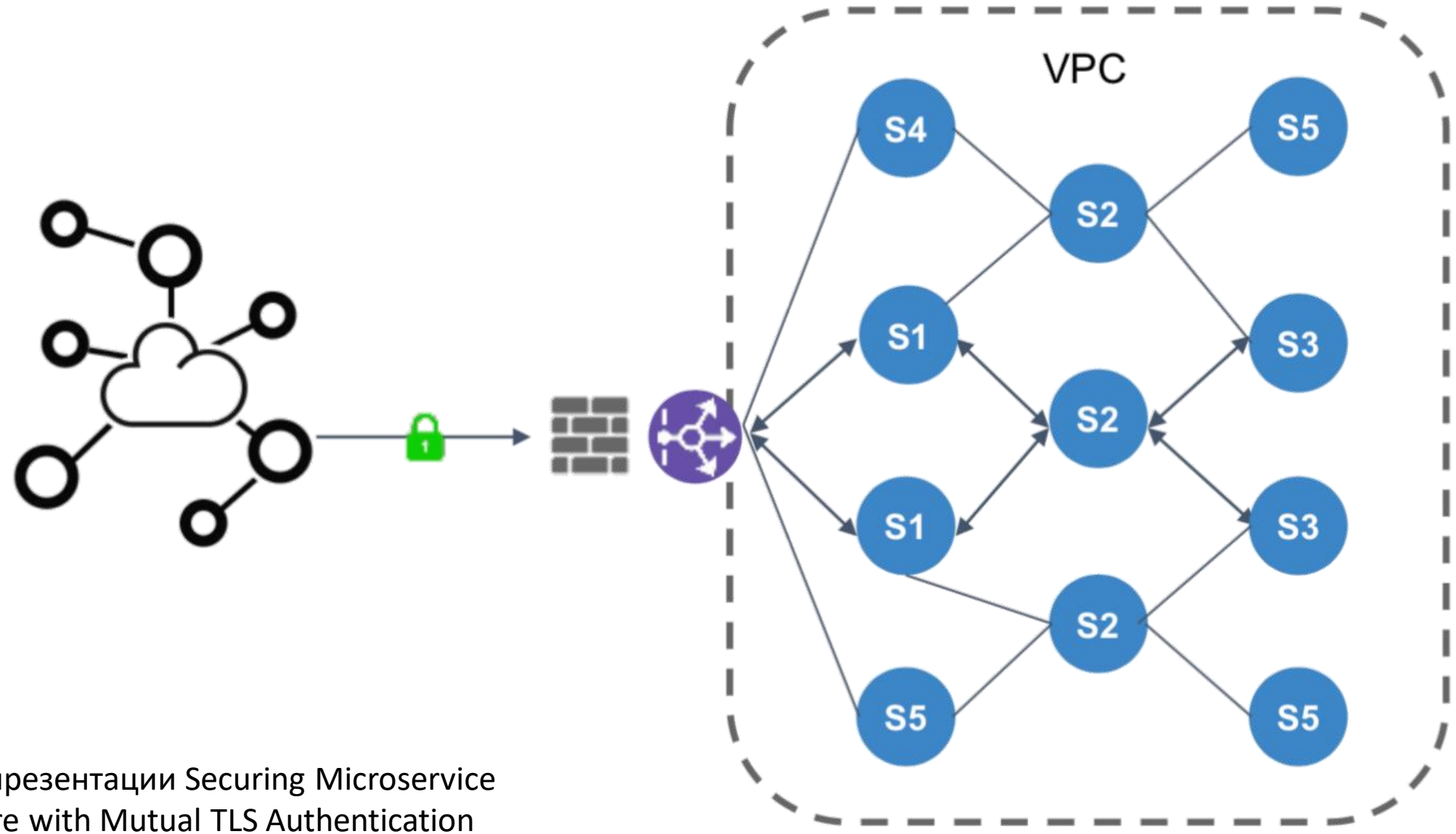


Схема из презентации Securing Microservice Architecture with Mutual TLS Authentication

Что хочется

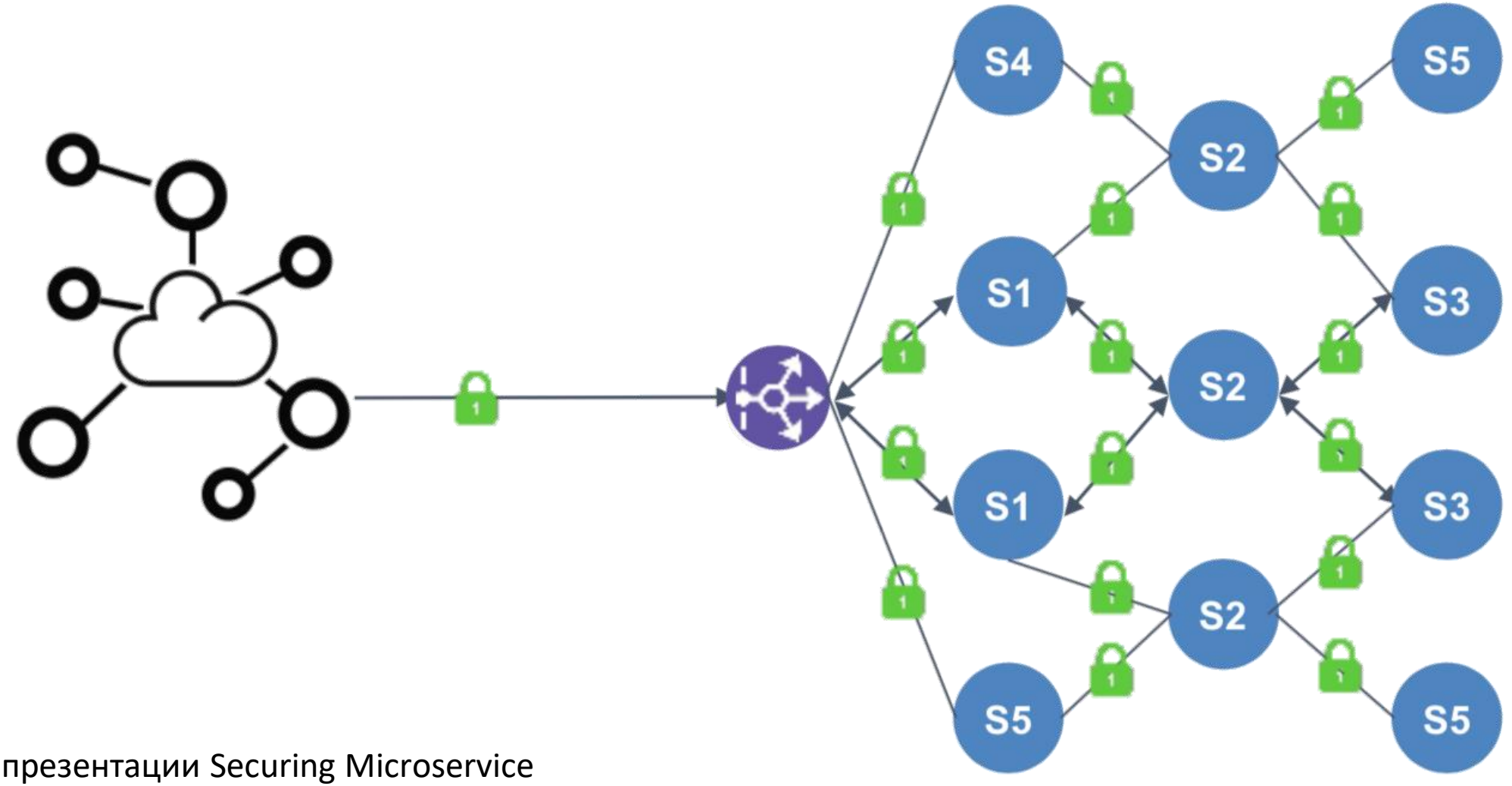
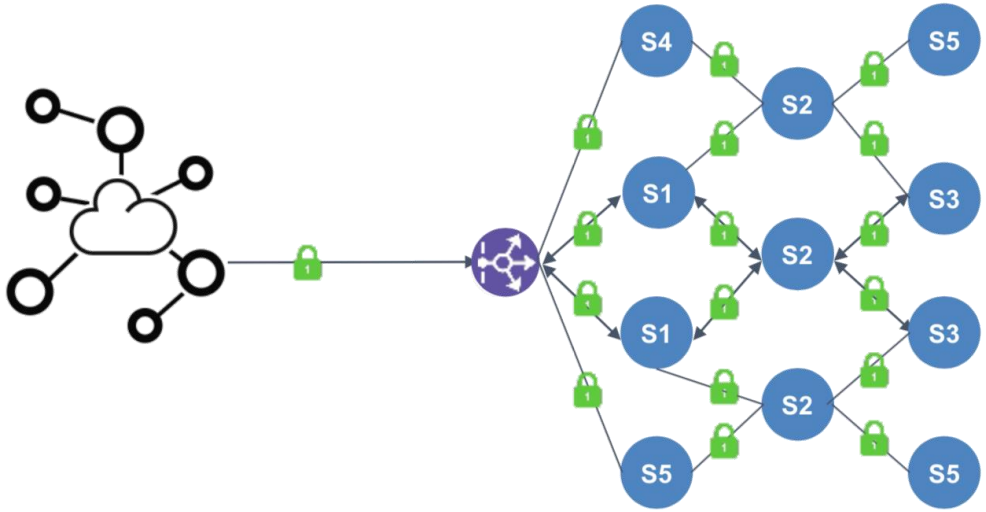


Схема из презентации Securing Microservice Architecture with Mutual TLS Authentication

Задачи



- Взаимная аутентификация микросервисов
- Защищенный транспорт
- Авторизация запросов к микросервисам

Схема из презентации Securing Microservice Architecture with Mutual TLS Authentication

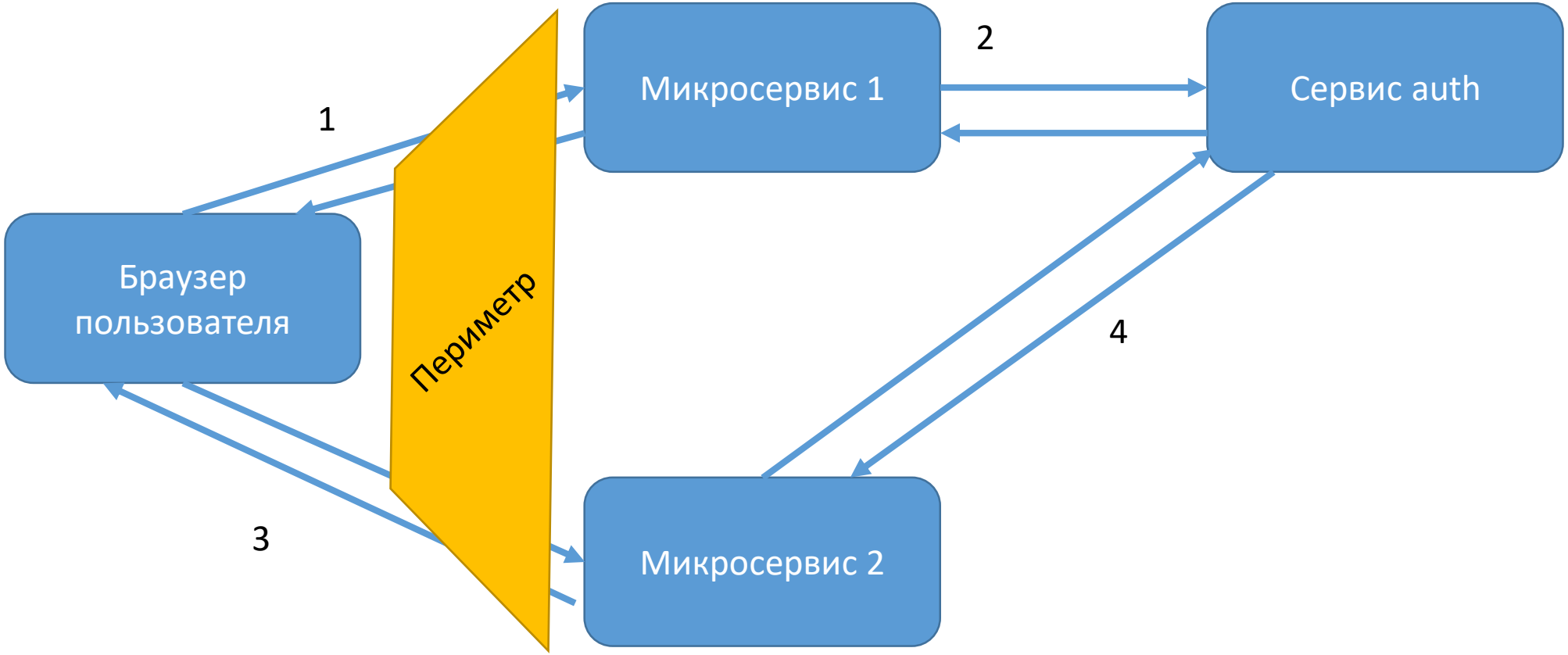
Взаимная аутентификация микросервисов

- Не нужна – мы в ~~домике~~ периметре
- На уровне инфраструктуры (ACL)
- С использованием общего секрета
  - токен в заголовке, параметре (cookie и http auth как частный случай)
  - нужно управление общими секретами (bootstraping/provisioning)
- С использованием ассиметричной криптографии (SSL-сертификаты)
  - нужен свой PKI (bootstraping/provisioning)
  - защищенный транспорт в виде бонуса
- Identity провайдер (внешний или внутренний)

Авторизация запросов к микросервисам

- Оставляем cookie во всех запросах, микросервис должен пойти к Security Manager и спросить, можно ли пользователю его вызвать
- Claims-based authorization
  - JWT
- Комбинированная схема
  - на клиент отдается session id
  - между микросервисами ходит токен с правами
  - gateway выполняет конвертацию

Вариант auth* 1



JSON Web Token

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

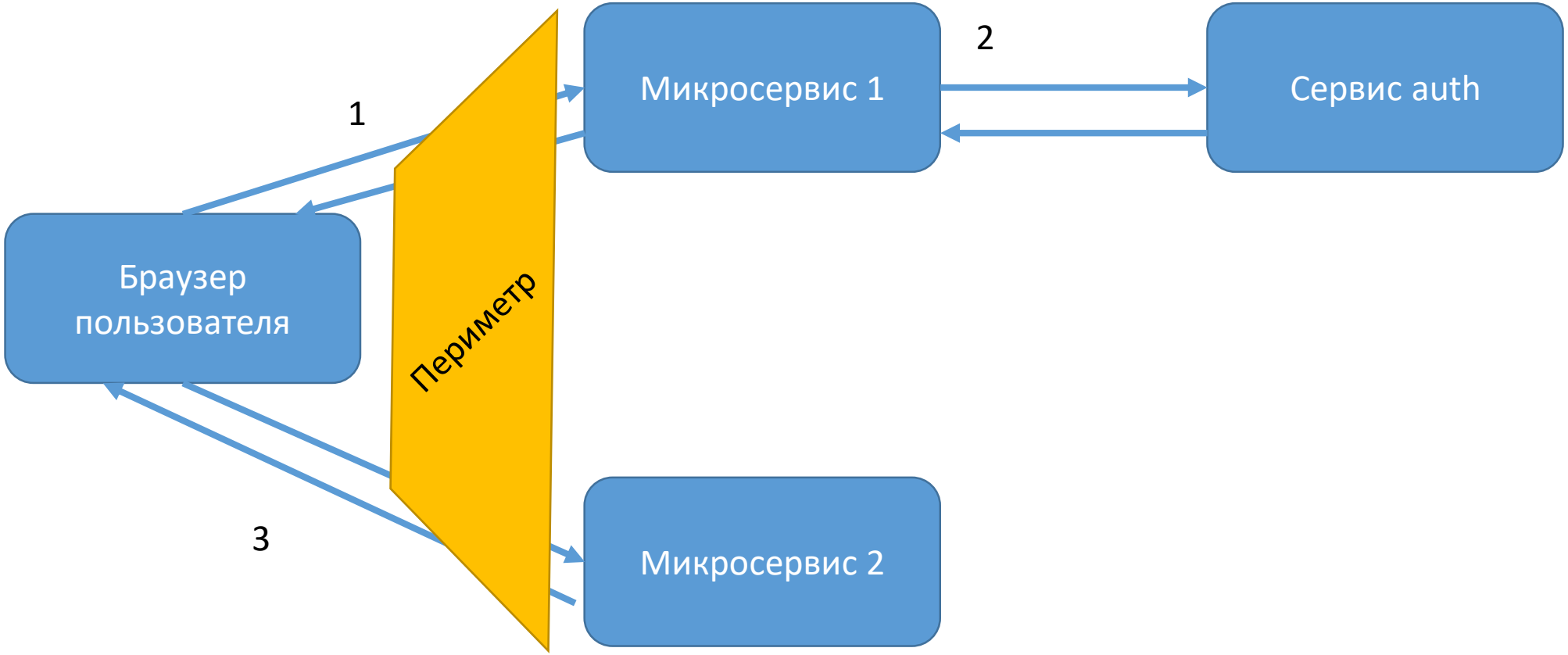
PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) secret base64 encoded
```

Вариант auth* 2



DEMO: JWT

Ваши вопросы?

Петухов Андрей
petand@seclab.cs.msu.su

Литература

- CSRF и cookie
  - https://habr.com/post/272187/
- CORS
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
  - https://portswigger.net/blog/exploiting-cors-misconfigurations-for-bitcoins-and-bounties
- Authc & Authz в микросервисах
  - https://jwt.io/
  - https://medium.com/tech-tajawal/microservice-authentication-and-authorization-solutions-e0e5e74b248a
  - https://www.slideshare.net/lmeirosu/mtls-securing-microservice-architecture-with-mutual-tls-authentication