OAuth 2.0 Vulnerability Impact Study

ISE6100 - Security Project Practicum

Authors: Brian Quick, brian.e.quick1@gmail.com
Russel Van Tuyl, Russel.VanTuyl@gmail.com
Sumesh Shivdas, msumeshfi@gmail.com

Advisor: Stephen Northcutt
Accepted: December 28th, 2016
Executive Summary

On 28 October 2016, the GIAC Enterprises CIO tasked a tiger team to answer concerns regarding a potential threat involving OAuth 2.0 and GIAC Enterprises applications. This analysis indicates that GIAC Enterprises currently owns and maintains 12 backend servers that contain vulnerabilities due to the implementation of OAuth 2.0. This vulnerability significantly impacts GIAC Enterprises. Covert remote attacks are possible from the Internet and create an unacceptable risk to GIAC Enterprises Fortune Cookie® intellectual property.

The test cases conducted by our team provided critical lessons on OAuth 2.0 traces and application calls which allowed the team to delineate how web applications in GIAC Enterprises were vulnerable. The tiger team recommends Option A, a solution that the GIAC Enterprises can implement as a fix with the least amount of effort and cost. The implementation will cost an estimated $17,150.00 with 343 man-hours required to fix the vulnerability. Web application programmers are available within the GIAC Enterprises IT department and can be utilized to help remediate this vulnerability.

1.1. Background

The tiger team was informed that GIAC Enterprises is a small to medium sized growing business with an estimated 1,000 employees, two data centers, and 200 people in central business and IT. GIAC Enterprises has a large number of 1099 based individual contractors that submit Fortune Cookie® sayings via a remote application and is the largest supplier of Fortune Cookie® sayings in the world. GIAC Enterprises tasked our team with identifying applications affected, vulnerability impact, OAuth 2.0 best practices, and to formulate recommendations to ensure GIAC applications are secure.
2. Initial Analysis

The chartered tiger team began the evaluation by performing an initial analysis on all GIAC Enterprises assets to identify which applications are affected by this vulnerability.

2.1. Vulnerability Analysis

In November 2016, students from The Chinese University of Hong Kong presented research on three vulnerabilities affecting the OAuth 2.0 framework. The identified vulnerabilities impacted the authorization and authentication aspects of the framework due to implementation flaws and not the framework itself (Yang, Lau, & Liu, n.d.). The vulnerabilities take place in how the mobile application’s backend server fails to validate untrusted input. A correctly positioned attacker can make user ID substitutions and gain unauthorized access.

Exploiting the identified vulnerabilities would allow a remote attacker to gain access to a GIAC Enterprises user account and potentially access sensitive data. The attacker will not need access to the target user’s mobile device and does not need the target users password to exploit the vulnerability. This attacker requires no interaction with the victim. An in-depth technical review of the exploit can be found in Appendix A. Additionally, the tiger team’s chronicles of understanding and identifying the exploit are found in Appendix C.

2.2. Asset Analysis

The tiger team found that 12 GIAC Enterprises backend servers were implementing OAuth 2.0 in a way that created a vulnerability.

2.3. Impact

The initial analysis has concluded that GIAC Enterprises owns a mobile application with an estimated 2,200 user accounts. Twelve backend servers supporting the mobile application implement the OAuth 2.0 framework in an insecure way. This

Brian Quick, Russel Van Tuyl, and Sumesh Shivdas
vulnerability significantly impacts GIAC Enterprises by allowing a covert remote attacker to access to Fortune Cookie® intellectual property.

2.4. **Best Practices**

GIAC Enterprises servers must not trust any public OAuth 2.0 based clients. The servers must confirm the validity of an authentication token directly with an authorization provider. The server must enforce Transport Layer Security (TLS) with strong cryptographic standards and HTTP Strict Transport Security (HSTS) headers. Additional information regarding best practices for the OAuth 2.0 framework are found in Appendix B.

3. **Remediation Options**

One remediation option, Option A, is for GIAC Enterprises to mandate that external contractors utilize OAuth 2.0 only, these contractors are also called 1099 due to their tax status. GIAC Enterprises will also limit the authorization providers to Google and Facebook only. Internal employees must authenticate using an existing and already implemented centralized identity management solution provisioned for internal employees or non-1099 contractors. This step will immediately limit exposure because 1099 contractors have limited access to the internal systems of GIAC Enterprises. GIAC Enterprises applications must be reprogrammed to verify and validate an authentication token with an authorization provider before granting access. Additionally, they will be configured to adequately check an authorization provider's signature when validated by OpenID Connect (OIDC). These top-tiered authorization providers have secure implementations and authorization servers located in the United States (U.S.), which will serve U.S. based 1099 contractors (Yang, 2016). Table 1 found in Appendix D shows the cost breakdown in both man-hours and dollars for this solution in greater detail.
A second remediation option, Option B, is to block access to all Internet-facing services and require workers to establish a VPN connection to access the resource server on the internal network until the vulnerability is remediated. This option will mandate GIAC Enterprises to provision the large number of 1099 contractors into their internal identity management systems and grant them access to internal systems. This option requires that the identity management server is upgraded (hardware and license) to handle new identities.

A phased approach is taken to prioritize the remediation of the vulnerability as outlined in Option A. After successful implementation in development, the changes are pushed to production. GIAC Enterprises’ security operations center can add signatures to existing web proxies and intrusion detection systems to identify multiple users authenticating from a single IP address. This step will afford additional measures for detecting and allowing business operations to continue as normal with no impact to operations.

4. Recommendation

The tiger team recommends Option A, to limit the OAuth 2.0 based authentication to the contractors in 1099 operational status. Internal employees must authenticate using a centralized identity management solution. This option will limit risk because contractors in 1099 status have limited access to the system. Authorization providers will be limited to two of the top tier identity providers, Google and Facebook which have secure implementations of authorization servers (Yang, 2016).

5. Conclusion

In this paper, we have identified the impact of the recently disclosed OAuth 2.0 framework vulnerability, the best practices, and options for GIAC Enterprises to mitigate the OAuth 2.0 vulnerability. Our research, planning, and approach show that GIAC Enterprises should choose Option A.

Brian Quick, Russel Van Tuyl, and Sumesh Shivdas
Brian Quick, Russel Van Tuyl, and Sumesh Shivdas
References


LinkedIn, Authenticating with OAuth 2.0 Retrieved December 10, 2016 from https://developer.linkedin.com/docs/oauth2


Brian Quick, Russel Van Tuyl, and Sumesh Shivdas


Appendix A - OAuth 2.0 Vulnerability

OAuth version 1.0 was published in June 2008 as Request For Comments (RFC) 5849 and is a protocol that was created “to solve the common problem of enabling delegated access to protected resources. (Hammer-Lahav, 2010)” The protocol was subsequently updated in October 2012 as The OAuth 2.0 Authorization Framework. OAuth utilizes the Hypertext Transfer Protocol (HTTP) to obtain authorization and access to a user’s account information maintained by a third party (Anicas, 2014).

RFC 6749 outlines four roles that are used within the framework: resource owner, resource server, client, and authorization server (Hardt, 2012). Additionally, there are four authorization grants: implicit, client credentials, resource owner password credentials, and authorization code. The implicit authorization grant was primarily designed “for clients implemented in a browser” (Hardt, 2012) and is frequently used in mobile applications (Anicas, 2014). As such, the implicit authorization grant will be the primary focus. Under the OAuth 2.0 framework, the GIAC Enterprises mobile application acts as a client, the GIAC Enterprises backend server acts as a resource server, the mobile device end-user acts as the resource owner, and a third party identity provider like Facebook or Google acts as both the resource and authorization server. The flow of operations documenting the GIAC Enterprises mobile application gaining access to a mobile user’s profile data is illustrated in the following image.
When the OAuth 2.0 framework is implemented in a secure manner, the mobile application’s backend server will validate the information it received from its own mobile application directly with the resource and authorization server as shown in steps six through nine in the figure above (Yang, Lau, & Liu, n.d.).

In November 2016, students from The Chinese University of Hong Kong presented their research on three vulnerabilities affecting the OAuth 2.0 framework. The identified vulnerabilities impacted the authorization and authentication aspects of the framework and are due to implementation flaws and not the framework itself (Yang, Lau, & Liu, n.d.). The vulnerabilities take place in how the mobile application’s backend server fails to validate untrusted input. The data is untrusted because it could be trivially modified.

The first vulnerability occurs when the mobile application’s backend server does not validate the access token with the identity provider and merely accepts the data received by its own mobile application as trusted and valid (Yang, Lau, & Liu, n.d.). A malicious mobile device user could have provided the mobile application’s backend server information.
server with a victim user’s account identifier and an invalid access token. Because the
access token is not validated by the mobile application’s backend server, the malicious
user is now logged into the application as the victim user.

The second vulnerability is also found in the mobile application’s backend server
and its implementation of the OpenID Connect (OIDC) protocol (Yang, Lau, & Liu,
n.d.). OIDC was created by Google and Facebook to reduce the number of round trips
shown in the previous image. Using OIDC, the identity provider returns a signed user
profile in steps two through five as shown in the image below. The vulnerability exists
when the mobile application’s backend server doesn’t check the signature associated with
the signed user profile prior to logging the user into the application in step six.

![Figure 2: OAuth 2.0 Application Flow Using OIDC](image)

The third vulnerability instance appears when the mobile application gathers the
user profile itself from the Android account management system or contacts the identity
provider directly, bypassing the identity provider’s client side application. Under this

Brian Quick, Russel Van Tuyl, and Sumesh Shivdas
scenario, an access token isn’t provided to the mobile application’s backend server and therefore it isn’t able to validate the information if OAuth was implemented in a secure manner.

In all instances, the vulnerability exists because the mobile application’s backend server doesn’t validate the received user information from its own mobile application. This validation is done by contacting the identity provider directly with the previously issued access token when using the OAuth 2.0 framework or by validating the signature associated with the previously provided user profile when using the OIDC protocol. The impact is such that a malicious user could conceivably log into the mobile application as any user and potentially access proprietary and sensitive information.

Additional research to identify a vulnerable application and review current implementation can be found in Appendix C.
Appendix B - OAuth 2.0 Best Practices

The OAuth 2.0 protocol is used by GIAC Enterprises mobile application running on iOS and Android devices. The backend web application is running on a Linux platform with an Apache Tomcat web server and an Oracle database. OAuth 2.0 defines the following four roles for the protocol

a. **Resource Owner** - This is normally the end-user or person who owns the resource. OAuth 2.0 requires this entity to authorize the delegation of privileges for the client or resource server to use on his/her behalf

b. **Resource Server** - This is the server hosting the protected resource for the resource owner. It is capable of responding to the protected resource requests using access tokens provided by the authorization server

c. **Client** - This is an application making a request to access the protected resource on behalf of the resource owner, with his/her authorization. This can be running on personal computers, mobile devices, servers, or other devices

d. **Authorization Server** - This is the server issuing the access token to the client after successfully authenticating the resource owner and obtaining his/her authorization

The best practices listed here are based on the defined roles. The recommendation here does not explicitly include some of the standard best practices such as peer code review, security integrated into the development cycle, intrusion tests, use of standard ciphers, static and dynamic code analysis. These practices are already incorporated and adhered to by GIAC Enterprises.
Resource Owner

The resource owners are GIAC Enterprises end-users and contractors who submit the Fortune Cookie® sayings using the GIAC Enterprises mobile application. The best practices for the resource owners are:
1. Protect the device with strong a PIN or preferably with biometrics if available
2. Prefer to use an identity provider which allows multi-factor authentication and not just a password
3. Use passphrases instead of passwords where allowed
4. Update the mobile device with the latest patches
5. Review and restrict the permissions to the installed applications to the minimum needed
6. Run a current version of antivirus and anti-malware software on the devices with automatic signature updates
7. Prefer third party applications which use external user-agents for OAuth (OAuth pattern) for authorization. Applications with embedded user-agent (like web view) can access sensitive information like user credential, content of the cookie, and can even modify web page contents. The access to the users credential by third party application defeats the purpose of the OAuth 2.0 framework, which is designed to eliminate the need of sharing the credentials.

Resource Server

The resource server is the GIAC Enterprises web application which processes the input provided by the employees and contractors using the GIAC Enterprises mobile applications. The best practices for the resource server are:
1. Harden the OS, web server, and database based on industry standard benchmarks
2. Grant access to the OS and database based on a need-to-know basis and limit the administrative privileges on the systems
3. The resource server must not trust its own client. The server must verify and validate the information received from the client with the identity provider’s server (Ping Identity, n.d.)
4. The resource provider server must mandate the access token. It must not grant access without verifying and confirming the access token with the identity provider’s server
5. If the identity provider signs the identity token then the resource provider must verify the signature before providing access to the resources
6. Mandate current version of TLS and strong cipher suites to protect the access token
7. The resource server must enforce HTTP Strict Transport Security (HSTS)
8. Resource servers must handle the access tokens as sensitive information and avoid storing it. If temporary storage is needed it should be encrypted (Kiani Khash, 2016)

Client

For GIAC Enterprises, there two types of clients, public and private.

Public Client

The public client is our GIAC Enterprises mobile application installed on an end user’s device. The best practices for the public client are:

1. Use Proof Key for Code Exchange (PKCE) to protect against inter-app URI-based communication sending data over insecure channels. This will prevent exposure of authorization response by eavesdropping or interception (Sakimura, 2015)
2. PKCE is not supported with OAuth implicit flow and hence the implicit flow authorization must not be used (Bradley, 2016)
3. The client must use the web browser as the external user-agent and must not use embedded user-agents for authorization (Bradley, 2016)

4. The client must not use “localhost” for loopback IP redirects. It must specifically use “127.0.0.1” (for IPv4) to prevent misconfigured routing or interferences by client-side firewalls. In case the loopback adapter is used, then the client must only open a port on the loopback adapter when starting the authorization request and must close it after the response is received.

5. To prevent cross-application request forgery, the client must use a strong and secure random number in the ‘state’ parameter for authorization request and must reject any incoming authorization response without a state value that matches the request (Bradley, 2016)

6. To prevent against authorization server mix-up, the client must register a unique redirect URI (different paths) for different authorization server it supports. The client must reject authorization responses if the redirect URI does not match the redirect URI in the pending outgoing authorization request (Fett Daniel, 2016)

7. Use the official release of the applicable Software Development Kit (SDK)

8. Do not use or distribute an application's secret used during registration in public clients (LinkedIn, n.d.)

9. The application must request the scope incrementally (principle of least privilege), where available (Google OAuth 2.0 implementation) and during the time the access is required rather than requesting all accesses up-front

10. Must avoid using the resource owner password credential grant type. If this grant type is unavoidable then the passwords must not be logged or stored and the scope requested must be minimal (Lodderstedt, 2013)
Private Client

GIAC Enterprises private client is installed on the Linux based system with restricted access. This is used to retrieve the data on behalf of marketing staff. The best practices for a private client are:

1. The shared secret must be protected. The system and components must be hardened using industry standard applicable security baselines
2. The server hosting the private client must be under strict access control and physical security
3. A process must be in place renew the shared secret in case of compromise

Authorization Server

GIAC Enterprises does not own or maintain an authorization server. Some of the best practices for an authorization server are:

1. The authorization server must mandate the user of PKCE from public native application clients
2. The server must be able to distinguish between public native application clients from the private web clients
3. The server should require the client to pre-register the complete redirection URI and must be mandated for the public native clients
4. For custom URI scheme based redirects, authorization servers should enforce that the clients use reverse domain based schemes
5. The authorization server must prompt the user for consent if the authenticity of the client cannot be assured
6. The authorization server must not rely on the public client identification using the shared secret. These secrets could be distributed to multiple users as a part of the application distribution
7. Must mandate latest recommended version of TLS with strong encryption
8. Authorization servers must mandate the registration of specific redirect URI, including path components. It must reject authorization requests which do not match the one registered by the client (Richer, 2015)

9. Must verify and validate all the parameters sent by a client

10. Must prevent against brute force attack against a client authentication using a password

11. Must set a quick expiry on the token and verify that expired tokens are not used

12. The authorization server must ensure that the tokens generated are unpredictable and cannot be generated by unauthorized parties

13. Authorization servers must generate short-lived and single-use authorization codes. Authorization servers must revoke all grants to access tokens if repeated attempts are detected to exchange authentication tokens with the same authorization code (Lodderstedt, 2013)

14. The authorization server must protect its end-points from Cross-Site Request Forgery (CSRF) attacks

15. The authorization server must have an interface to blacklist a client or a specific version of a client (concept like certificate revocation)

16. Must mandate latest recommended version of TLS with strong encryption

17. Authorization servers must validate the client registration requests to prevent fake registrations (Richer, 2015)

18. The authorization server must not disclose too much information about an inactive token, this could reveal the state to the attacker (Richer, 2015)
Appendix C - OAuth 2.0 Vulnerability Research

Understanding the Vulnerability

Notes taken when reviewing the white paper found at

- Vulnerability is due to implementation, not the protocol itself
- The protocol doesn’t provide or recommend secure usage.
- Leverages untrusted data
- Requires no user interaction or MiTM to exploit
- User will receive no indication of malicious behavior
- OAuth not designed for mobile apps

Mobile App Flow

1. User opens Android’s GIAC mobile app
2. GIAC mobile app connects to Android’s FaceBook mobile app (the identity provider)
3. FaceBook mobile app sends request to FaceBook backend servers on the Internet
4. FaceBook backend server sends an Access Token (AT) and optional user profile back to the FaceBook mobile app
5. The FaceBook mobile app sends the AT and profile to the GIAC mobile app
6. GIAC mobile app sends the AT to the GIAC backend servers on the Internet
7. GIAC backend server should talk to Facebook backend server to validate AT
8. Facebook backend server sends GIAC backend server the user’s information

OpenID Connect protocol (OIDC) is used to reduce round trips

Brian Quick, Russel Van Tuyl, and Sumesh Shivdas
- Immediately sends back user data, cutting out the communication between the GIAC backend servers and the Facebook backend servers.
- Implements User Profile Signing
- GIAC Backend server verifies user based on signature without having to contact IDentity Provider (IDP)

Vuln 1: GIAC backend server doesn’t validate authenticity of data received by Facebook mobile app and logs the user in without validating the token (skipping steps 6-8 above);
Vuln 2: When OIDC is used, GIAC backend server takes signed user data as trusted and doesn’t validate the signature as it should (skipping steps 6-8 above);
Vuln 3: GIAC mobile app uses local user account information instead of the Access Token provided by IDP; ANDROID ACCOUNT MANAGER

WORST CASE IMPACT:
An attacker can log into the GIAC application as a privileged user and access intellectual property that is authorized to be viewed or updated by that user without the user’s knowledge or participation.

Initial research identified that 99 of the top 400 mobile applications using Google and Facebook as an IDP were vulnerable.

Identifying the Vulnerability with First Hand Research
The tiger team worked to identify any vulnerable applications still available as a method to thoroughly understand the vulnerability.

Test Environment - Team Member A
The tiger team spoke with a mobile application penetration tester from Sword & Shield Enterprise Security about the best toolset to use. He recommended using

Brian Quick, Russel Van Tuyl, and Sumesh Shivdas
Genymotion as a test bed. Team member A downloaded Genymotion but found it utilizes Virtual Box; he already had Hyper-V enabled and you can’t run multiple hypervisors on a single machine.

December 15, 2016

Team member A searched for a Hyper-V capable Android test suite. Came up with Visual Studio and its old Xamarin Android emulator. Team member A spun up an Android Marshmallow (version 6) virtual machine and everything worked great. He spent the next 3 hours trying to get Google Apps installed to be able to access the Google Play Store. He wasn’t able to get it installed.

December 16, 2016

Team member A spun up an Android Lollipop (version 5) virtual machine. The machine wouldn’t start due to some graphics display problem. Team member A then downloaded the Android Nougat (version 7) SDK from Google. Visual Studio doesn’t have an emulator for it yet. He then spun up an Android KitKat (version 4) virtual machine. Team member A was finally able to get Google Apps installed and was able to log in with a development account, russel.sans.dev for further testing. He also installed a Burp Suite certificate to man-in-the-middle (MiTM) traffic from the VM through Burp’s proxy. He downloaded several apps looking for one that uses OAuth 2.0.

Test Environment - Team Member B

December 19 – December 20, 2016

Team member B downloaded and installed Android Studio with the goal to look at some of the sample published OAuth 2.0 implementation, compile the sample OAuth 2.0 code, and see it in action. Downloaded and installed Burp Suite to review communication between various components. Compiled a simple login demo authentication program. The following images illustrate installing the custom application and its login screen.

Brian Quick, Russel Van Tuyl, and Sumesh Shivdas
Figure 1: Sample ISE 6100 Application
Figure 2: Sample ISE 6100 Application Login Screen
Compilation of the sample OAuth 2.0 was unsuccessful, due to some mandatory files missing in the installation, did not pursue further due to limited time availability.

December 20-21, 2016

Team member B created emulated version of Android and tried using the web version of OAuth 2.0 and review it via Burp. The initial attempts failed as the latest version of emulator packaged with the Chrome web browser does not allow use of Burp.
as a MiTM proxy. It enforces HTTP Strict Transport Security (HSTS) and does not allow the use of the Burp Certificate Authority (CA) certificate.

![Figure 4: Burp CA Certificate Installed]

![Figure 5: Burp CA Certificate not Working – Chrome Enforcing HSTS]

Brian Quick, Russel Van Tuyl, and Sumesh Shivdas
December 22-24, 2016

Team member B created an old version of Android, which uses an older browser that does not enforce HSTS. With this setup, it was possible to review some of the communication for web applications using HSTS. Attempt to install native application (Twitter, Quora, Khan Academy) all failed with the error “You don’t have any device”, could not resolve this issue with the android studio emulator and had to drop the idea of simulating a full flow capture of the flow involving a OAuth 2.0 client and identity provider installed on the android emulator.

Android Studio Virtual Devices (emulators) do not have Google Play Store installed. Attempted to install Google Play Store and Twitter application. The setup did not last long and had to be deleted due to an adware getting installed as a part of one of the application, would have been good digging for reverse engineering malware project.
Burp snippet from malware communication, virustotal screenshots indicating the site is a reported malware site. The message “13 viruses detected” and “the virus will damage your SIM card” are static messages displayed by the javascript (without any scanning of files), no antivirus software was running on the device.

```javascript
var __redirect_to2 = 'http://10X.YYY.ZZ.247/go.php?s=21508297177';
...

var currentTime = new Date();
var dayNames = new Array("Sunday", "Monday", "Tuesday", 
"Wednesday", "Thursday", "Friday", "Saturday");
var date = currentTime.getDate();
var year = currentTime.getFullYear();
document.write(dayNames[currentTime.getDay()] + " " + date + ", " + 
year);
</script>
</div>
<h1 id="headTxt2"><span id="blink">13 viruses detected</span></h1>
<div id="bodytxt">
<img src="data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAOUAAADpCAMAAADs1QBfAAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAAAAyJpVFh0WE1MOmNvbS5hZG9iZS54bXAAAAAAAD..."

<b>If this is not resolved in a few minutes, the virus will damage your SIM card</b>
<br>Step 1: Click on the install button and install "DU Cleaner" application
<br>Step 2: Open the application and click "CLEAN" button, viruses will be deleted automatically
<br>Step 3: Do not remove the application or viruses may become active again
</div>
```
Team member B attempted to install a new emulator, Visual Studio. It seemed to be very slow and old for the Windows 7 version and hence had to stop reviewing further.

**Mobile Application Testing**

December 17, 2016

Team member A created a second account to facilitate testing so that way he could attempt to steal its data (sumesh.sans.dev). Team member A pulled his personal Google+ profile for its unique identifier. This identifier is used to perform the attack that
exploits the OAuth 2.0 implementation vulnerability. The following identifiers were also retrieved:

- https://plus.google.com/11482576301588096386 (Russel Van Tuyl)
- https://plus.google.com/11056314511036267506 (Sumesh SANS Dev)
- https://plus.google.com/10827529004204689883 (Russel SANS Dev)

POST /auth HTTP/1.1
device: 3d529c3fe213043b
app: com.google.android.gms
Accept-Encoding: gzip
User-Agent: GoogleAuth/1.4 (donatello KTUB4P); gzip
Content-Length: 572
content-type: application/x-www-form-urlencoded
Host: android.clients.google.com
Connection: close

androidId=3d529c3fe213043b&lang=en_US&google_play_services_version=100890700&device_country=
us&app=com.google.android.gms&callerSig=389f18a451d07199354f2mb1s205ec65f2ced5700&client_sig=389f18a451d0719
915429193e05e65f2ced5700&token_request_options=CAk4QAQ1D139&Email=russel.sans.dev4@gmail.com&service=oauth2&https://plus.google.com/11482576301588096386

Figure 9: Authorization Grant Request for Google+

HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: Mon, 01 Jan 1990 00:00:00 GMT
Date: Sat, 17 Dec 2016 20:32:25 GMT
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Server: GSE
Alt-Svc: clear
Connection: close
Content-Length: 215

Auth=ya29.C123A09XY52Us0c0c3jL2QCqW08KlIUNPBQVHibQ03ajy4Fw_F-BrZeD42Gpqlr1ISwWS01--0Hsk25Z_fzcuDv6kEdO1gg7BLi4
QDms01suOfE7KSpLV14WnrXWtjo
issue=1&device=auto
Expires=1455201705
storeConsentRemotely=0
isTokenSnackBar=0

Figure 10: A Response Containing a Token

Request to https://android.clients.google.com containing the Wish mobile application package identifier.

Brian Quick, Russel Van Tuyl, and Sumesh Shivdas
December 18, 2016

Continued evaluating application for OAuth 2.0 implementation vulnerability.

The Wish iPad mobile application appeared to be vulnerable, but team member A wasn’t able to identify any calls from the application to its backend server. Team member A speculated that it might be problematic using Google accounts because the Android operating system is used on top of them. One of the other identity providers might be the weaker targets. However, the Google unique identifier is easy to find from a Google+ profile while the Facebook identifier is unique each time.

Identified the Planner Pro app, it uses Facebook, Google+, and Twitter as identity providers. Team member A was also able to identify calls to the Planner Pro backend web server. Even though the profile information was tampered with, the Planner Pro app still uses the original userid. Khan Academy does the same, returns the original account.

December 19th, 2016

Despite knowing which requests to modify, a successful attack hasn’t been performed.

The initial whitepaper was re-read and surfaced additional things that were overlooked.

1. No IDP applications were installed on the emulator

Brian Quick, Russel Van Tuyl, and Sunmesh Shivdas
2. A user account on the vulnerable service from the 3rd party needs to exist prior to the testing
   a. “While our current attack is demonstrated over the Android platform, the exploit itself is platform-agnostic: any iOS or Android user of the vulnerable mobile app is affected as long as he/ she has used the OAuth2.0-based SSO service with the app before.” (Yang, Lau, & Liu, n.d.)
3. Facebook was used as the target IDP in the whitepaper
4. The popular IDPs have been notified and may have potentially implemented changes to reduce the attack effectiveness.

A table was created to keep a record of testing user accounts and the identifiers associated to them per IDP.

<table>
<thead>
<tr>
<th>Username</th>
<th>Twitter ID</th>
<th>Google ID</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:Russel.Sans.Dev@gmail.com">Russel.Sans.Dev@gmail.com</a></td>
<td>811044775705149440</td>
<td>114852763015880960386</td>
</tr>
<tr>
<td><a href="mailto:Sumesh.Sans.Dev@gmail.com">Sumesh.Sans.Dev@gmail.com</a></td>
<td>---</td>
<td>110556314511036267506</td>
</tr>
<tr>
<td><a href="mailto:Brian.Sans.Dev@gmail.com">Brian.Sans.Dev@gmail.com</a></td>
<td>811041482991931392</td>
<td>108275329004204689883</td>
</tr>
</tbody>
</table>

*Table 1: Testing User Accounts and Identity Provider IDs*

Team member A attempted to perform the attack against the Planner Pro application using Twitter as the IDP. This is unique because the original research didn’t indicate an evaluation against this provider. This time, he created two user accounts (Brian.Sans.Dev & Russel.Sans.Dev) prior to performing the attack. Team member A noticed a web response from the 3rd party backend server to its mobile app that seems to indicate it validated the data correctly.
Team member A also found out that the steps to login to Planner Pro while creating an account were different than when trying to log in after an account had previously been created. The main difference was that the second time around, he was prompted to authenticate to Twitter using the web browser or the Twitter app installed on the emulator. He attempted to authenticate with the browser but the Twitter account setup for testing had been locked out due to automated activity. Team member A had to authenticate the test user accounts by providing his phone number to Twitter.

December 21, 2016

Team member A modified the request (see image below) from Twitter back to Planner Pro by changing the Twitter ID. The string ‘811044775705149440’ in the first image is the tampered data and represents the Russel.Sans.Dev@gmail.com user account. The original request had the string ‘811041482991931392’ representing the Brian.Sans.Dev@gmail.com user account.

Brian Quick, Russel Van Tuyl, and Sumesh Shivdas
GIAC Enterprises OAuth 2.0

Brian Quick, Russel Van Tuyl, and Sumesh Shivdas

Figure 13: Tampered Response from Twitter to the Planner Pro Application

POST /pages/users HTTP/1.1
X-Parke-Application-Id: 49b514b9d7689f4db5e1e62eb56e87d1
X-Parke-App-Display-Version: 6.3.4
X-Parke-Installation-Id: 201007c96-c063-4c07-b7c0-0be4fe89569da
X-Parke-OS-Version: 4.4.4
User-Agent: Parke Android SDK 1.1.1 (com.appy.planner/18) API Level 16
X-Parke-Client-Key: 71FD498f4C4L9xt2095Qw35q9S71jF2VqE4V
X-Parke-Client-Version: a1.11.1
X-Parke-App-Build-Version: 30
Content-Type: application/json
Host: plannerpro-master.us-east-1.amazonaws.com
Connection: close
Accept-Encoding: gzip
Content-Length: 311

{"authData":{"twitter":{"id":"811044775750149440","consumer_secret":"ocBuV5LQ5bozQ2L1gmrDAvYeXUxXtInlE2W1wWeBk4KCy3YW4LJ71","access_token":"5nL0R87yqetMeUm4bVw6cC32F","auth_token_secret":"TVWFZ10JD1uygNw20chkXnVPRRE3j5P0dALqyYa"}}}

Figure 14: POST from Planner Pro Mobile Application to its Backend Server Using Poisoned Data

The image below is a response from the Planner Pro backend server. The response contains an error message that indicates the Planner Pro backend server checked with Twitter to validate the user account but found it to be invalid. This is true because the data was previously tampered with. It appears that the Planner Pro application has implemented the OAuth 2.0 framework correctly by validating the access token with Twitter prior to logging the user in.
Figure 15: Authentication Failure Message from Planner Pro Backend Server

Team member A made a request to the Planner Pro backend server using the original untampered data. The Twitter ID was changed back to the one the access token was assigned to. The image below shows a successful request and response. Take note that response contains the user’s data.

Figure 16: Successful Authentication to Planner Pro

Team member A tested out the Drona Android mobile application. It looks like it uses OAuth, but it never sends any data to its backend. When a user logs out of an account, clears the app data, and logs back in, none of their “bookmarks” are restored. All data and user account information are held on the local device. No further testing on this application was performed.

Team member B identified a successful authentication request from Quora.com to Google, an identity provider, as shown below. This request was generated from using a web browser on Android emulator and through the use of the Quora mobile application.
GET
/google_/callback?state=eyJwZXJtcyI6IiIsIiwiYXNzd2Z0IiwgImlubmVudCI6IiwiaHR0cHM6Ly94dXMvcGhvbmUvaHRtbC8iLCJ3aWR0aDoiMTI4NzY5ODA2OSIsImNsaWVudCI6IiwiZm9yZSI6IiwiZGVmYXVsdCIsIiwiZnVuY3Rpb25JZiI6IiJ9&client_id=917071888555.apps.googleusercontent.com&redirect_uri=https%3A%2F%2Fwww.quora.com%2Foauth_window&response_type=code&scope=email+profile+https%3A%2F%2Fwww.google.com%2Fm8%2Ffeeds&approval_prompt=auto&access_token=ya29.Ci6AygDe-XPXYXYXABACACABCACMCN55vZUPfMKy1Z9w05w9KLJa_cYzTU_cVv&force_dialog=1&success=True&provider=google

The text below identifies a successful request to Google and highlights the returned access token that Quora will subsequently use.

The resource was found at <a href="https://www.quora.com/?access_token=ya29.Cl-6AygDe-XPXYXYXABACACABCACMCN55vZUPfMKy1Z9w05w9KLJa_cYzTU_cVv&force_dialog=1&success=True&provider=google">https://www.quora.com/?access_token=ya29.Cl-6AygDe-XPXYXYXABACACABCACMCN55vZUPfMKy1Z9w05w9KLJa_cYzTU_cVv</a>
Additional OAuth 2.0 Attacks

http://intothesymmetry.blogspot.ch/2014/02/oauth-2-attacks-and-bug-bounties.html
http://homakov.blogspot.ch/2014/02/how-i-hacked-github-again.html
http://homakov.blogspot.ch/2012/07/saferweb-most-common-oauth2.html
https://blog.srcclr.com/spring-social-core-vulnerability-disclosure/
https://www.synack.com/2015/10/08/how-i-hacked-hotmail/
http://intothesymmetry.blogspot.ch/2014/12/cross-site-request-forgery-ingithub.html
http://intothesymmetry.blogspot.it/2015/10/on-oauth-token-hijacksfor-fun-and.html
## Appendix D - Manpower Cost Breakdown

**Table 1. Cost Breakdown**

<table>
<thead>
<tr>
<th>Task</th>
<th>Development and Quality Assurance Cost (Man-hours)</th>
<th>One-time Install (Man-hours)</th>
<th>Phase</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable OAuth 2.0 for employees (privileged users). Update account provisioning workflow to allow OAuth 2.0 only for 1099-contractors</td>
<td>90 (2 Developers + 1 QA analyst) ($4,500.00)</td>
<td>120 (4 System admins) ($6,000.00)</td>
<td>I (deployment in a week)</td>
<td>Eliminate exposure of OAuth 2.0 vulnerability to privileged GIAC employee accounts</td>
</tr>
<tr>
<td>Verify authentication token with authentication provider and check signature if available</td>
<td>50 (2 Developers + 1 QA Analyst) ($2,500.00)</td>
<td>6 (2 System admins) ($300.00)</td>
<td>II (deployment in 3 weeks)</td>
<td>Limit exposure of OAuth 2.0 vulnerability to 1099-contractors</td>
</tr>
<tr>
<td>Limit authorization providers</td>
<td>35 (2 Developers + 1 QA analyst) ($1,750.00)</td>
<td>6 (2 System admins) ($300.00)</td>
<td>II (deployment in 3 weeks)</td>
<td>Limit exposure from insecure authorization providers</td>
</tr>
<tr>
<td>Retrofit client app to limit authorization provider</td>
<td>30 (1 Developer + 1 QA analyst) ($1,500.00)</td>
<td>6 (Client deployment) ($300.00)</td>
<td>III - Deployment can be planned at a</td>
<td>Update client to support only selected authorization</td>
</tr>
</tbody>
</table>

Brian Quick, Russel Van Tuyl, and Sumesh Shivdas
<table>
<thead>
<tr>
<th>support</th>
<th>later stage</th>
<th>providers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total cost</td>
<td>343 Man-hours. $17,150.00. No CapEx</td>
<td></td>
</tr>
<tr>
<td>Estimated elapsed time for server side deployment</td>
<td>3 weeks</td>
<td></td>
</tr>
</tbody>
</table>

The deployment is planned in a phased approach. This enables us to prioritize and address the most vulnerable area in the first phase, thus minimizing the risk.