Testimony to the ONC API Task Force

Greg Brail, Apigee

January 26, 2016

Overview

Thank you for giving us the opportunity to address the community regarding the opportunities and issues presented by the adoption of APIs.

In the last 10 years, we have seen the idea of a "web API" grow from an experiment created by a small number of web companies, to a technique used by popular social network mobile applications, to a mainstream set of technologies and best practices that are being used across the industry to make it easier for software developers to access data and services.

The very first web APIs may have been used for non-sensitive information such as weather forecasts and maps, but that quickly changed. Today we see APIs being used for everything from mobile payments to healthcare to wholesale financial services.

An API, at the simplest level, is a contract. The contract specifies how a software developer accesses an API, and tells the developer what to expect. A well-designed API makes this contract very clear through documentation and specifications that describe not only what kinds of requests the API expects, but what kind of security controls have been put in place and what set of security credentials a developer must acquire before she or he builds an application that uses the API.

Because an API is a contract, it is possible for the organization that offers the API to completely document and understand the interaction between the application that uses the API and the API itself. A variety of tools and techniques are available, both from commercial software vendors as well as from the open-source community, which can be used to ensure that API access is not allowed unless the client follows the contract. These tools may also be used to monitor API usage and gather data to understand exactly who is using the API and how.

This contract-driven interaction model makes it possible for the organization that provides an API to add policies and security controls on every interaction. An API team can therefore regulate which applications and end users are authorized to use an API and which parts of the API they are allowed to use. The team can also control what an authorized user can do, including limits on the number of API calls that can be made, or when they can be made. Finally, the team can follow the trail of API calls to understand exactly what authorized API users did, and what unauthorized attempts may have been made.

As a result, APIs, rather than being a new security risk, provide a well-documented, popular way for organizations to share access to data and services with third parties, while maintaining strict security

controls. Especially when compared to other ways of sharing this data, such as via web site, file transfer, email, or even printing press, a well-implemented API offers a stronger set of security controls.

Questions from the Committee

1. Are there any well-known threats or vulnerabilities associated with APIs themselves that should be addressed (e.g. security engineering considerations/best practices)?

Today, web APIs are a mature and popular technology choice. As such, there are a variety of security best practices that API providers should follow, and a great deal of information on these topics is available from various books and blogs.

Some of these security best practices are similar to those used on the World Wide Web today, such as proper use of TLS to ensure that communications are properly encrypted, and attention to common vulnerabilities such as “SQL injection” attacks.

Other best practices are specific to the world of APIs. For instance, it is common for the best-designed APIs to use a system of quotas and rate limits to protect the API against excessive traffic, even if that traffic comes from a properly authorized application.

a. As APIs are gaining adoption, are there steps organizations need to take to mitigate any additional threat vectors to data?

Of course, this is important especially when an API is being deployed that gives access to data that was not previously available via some other channel.

In these cases, a layered approach is best. For instance:

- Use existing network security best practices such as firewalls, intrusion detection sensors, routers, network design, and proper management of TLS to protect assets
- Use rate limiting mechanisms to control the amount of API data that may be consumed from all users, even authorized users
- Verify the identity of the application that consumes the API, as well as verifying the identity of the end user
- Scan any incoming data for attack vectors such as SQL injection and other malformed input that may be designed to cause a target system to crash

These are all security best practices that are specific to APIs.

However, in many cases APIs are being offered not as a way to expose data that has never been exposed before, but as a new way to get access to data that was previously exposed using some other mechanism such as a web site, a file transfer service, or even email.

In these cases, we would argue that moving to an API approach from one of these technologies results in a system that is actually more secure than the alternatives.

For instance, web sites are subject to any number of "screen scraping" techniques. Any reasonably clever web developer can use these techniques to discover what data is available to the web app. Since a web app is a complex application with many entry points, it is often the case that a clever developer can figure out ways to get access to data that they might not be able to access merely through a web browser. And even if a web site is 100 percent bug-free and this kind of access is not possible, when developers "screen scrape" the web site in this way, the team that controls the site does not have a good mechanism to understand if it is happening, who is doing it, what they are doing, and how much data they have accessed.

A properly secured API, however, can include security techniques such as the ones described above that can ensure that only properly authorized applications, built by authorized developers, can access only the data that is exposed by the API. Furthermore, the team providing the API can put traffic management features in place to control how much data each developer is allowed to access. And in the worst case, since the API was designed from scratch for easy access by developers, it is usually easier to use audit trails and logs to see exactly what was accessed, by whom, and for how long.

b. Are these just specific to APIs in general? What might be unique/specific to healthcare?

These security techniques apply to all APIs.

Of course, since the protection of personal information is especially important in the healthcare context, it is important to pay special attention to issues such as data encryption (both at rest and in transit). It is also important to design mechanisms that ensure data is only shared when the patient has given consent, and to give the patient tools to understand what consent was granted and to withdraw that consent when necessary.

c. How does the issuer of the API ensure that the API won't become a tool used for malicious activity which could compromise the data source?

The first lines of defense against malicious activity are the ones described above—ensuring that each API call comes only from a properly-authorized application, on behalf of a legitimate user, and that both application and user are only accessing the data that they are authorized to access, and that neither is accessing the API more than they have agreed to in their contract.

Of course, with any technology it is always possible for a legitimate user to exceed the bounds of their authority. For instance, in the API world a developer could sign up to build a legitimate app, but then over time turn their app into a "bot" that attempts to gather large amounts of data from the API, perhaps for competitive purposes.

When that happens, it is critical that the API maintain an audit trail that shows which API calls were accessed, by whom, and when. More advanced technologies are becoming available today that can scan such an audit trail to detect "bot-like" activity and stop it before it goes too far.

2. How are APIs distributed in a way that the recipient/end-user of the API can trust the API is authentic?

Similar to web applications on the Internet, any API that handles sensitive data must only be accessed using TLS (Transport-Level Security, sometimes referred to as "SSL") to encrypt the information, and to ensure the identity of the API server itself. That way the client calling the API can ensure that it will not attempt to use any API server that does not present the correct digital signature.

Furthermore, a well-run API will offer a complete "web portal" which allows the developer considering the use of the API to learn more about the API and interact with the team that offers it. An API with a good portal and web presence, and better yet one that is backed by an enthusiastic community of developers who take pride in their work and answer questions from users and potential users, is much more attractive than an API that is just an empty storefront. A savvy developer will think twice before building mission-critical infrastructure on an API that appears to be abandoned by its creators, but the same developer will enthusiastically trust an API that is backed by an engaged community.

3. Are there existing metrics or is there a need to develop metrics to measure the maturity of security and privacy controls in the use of APIs?

While there are many well-known security best practices for APIs, there is not necessarily an "API security score" or something similar that the industry has agreed on. Such a "score" could certainly be developed as part of this effort, however.

4. Is there a catalogue or store of tools that are built for the APIs for third parties to access?

APIs themselves are typically discovered using the World Wide Web, just like so many things. A well-designed API will offer a "portal" that allows developers to discover what the API does, read documentation, and sign up for access, all via self-service. Depending on the security posture of the API, an API may allow access to any developer who asks, or require approval from a member of the API team, or even be set up so that only authenticated users can access the portal to learn about the API at all.

A good API will offer on this portal not only information and documentation, but tools to make it easier to use the API. For instance the most popular APIs offer a set of SDKs (Software Development Kits) for various platforms and languages that make it easier to use the API.

With that said, the beauty of APIs is in their simplicity. All of the best APIs may be used by any programmer, from any environment, as long as they possess an application that is able to use the HTTP protocol and parse JSON data -- in other words, just about every computing device in use today.

Furthermore, because of this simplicity, the developer who decides to adopt an API in an application can see exactly what data they are sending to the API -- it is not hidden by some proprietary protocol or opaque library. This improves the security posture of the application that uses the API, because nothing is left hidden.

5. Are there known compliance implications with the use of APIs?

There are no compliance implications to the use of APIs that do not already apply to data that is exposed via a web application or file transfer system. Industry standards such as PCI and HIPAA apply to APIs just as they apply to other systems. Similarly, information privacy or data protection laws such as ECPA in the

US, PIPEDA in Canada, ECHR in Europe, and DPA in the UK may also have implications with the use of APIs.

6. What are the perceived and actual security concerns or barriers to the adoption of APIs?

The actual security concerns are those that we have previously discussed: ensuring that only authorized end users and applications access APIs, controlling the amount of API traffic that they are allowed to generate, ensuring that the API traffic does not contain malicious content, and auditing all the traffic for later analysis and risk mitigation. These best practices are critical, but at the same time they are well known and there are many products available from commercial firms as well as open-source organizations that can help.

The perceived security barriers are often higher. Many people believe that by "offering an API," an organization is suddenly "exposing data" that was previously locked away in a secure vault. While this sometimes happens, it is more common that the data being offered via API was previously transferred between systems using a marginally-secure file transfer system, or displayed by a complex web application, or even routinely sent via e-mail.

APIs are different because they are designed from the ground up to do only one thing, and that is to provide programmatic access to developers who code applications. As such, an API provides a much tighter contract than a web application. There is no reason why an API built to today's best practices cannot be set up so that every single API interaction is authenticated, authorized, encrypted, and audited to ensure that no data protection policies are breached.

7. How can these risks be mitigated/how are you addressing this?

Every API provider is responsible for following API security best practices such as those described earlier in this document.

To that end, we at Apigee provide a software product that makes it possible to implement these security practices, by doing things like authenticating API traffic, controlling the amount of traffic from each application, scanning for threats, and detecting and rejecting "bot-like" activity automatically.

Furthermore, there is a great deal of information to be found in various webinars, blogs, books, and other formats written by experts from Apigee and from many other companies that describes how APIs can be deployed that allow developers to quickly build new solutions, while protecting against security risks.