FX Labs Inc - API Cybersecurity Company
https://fxlabs.io/

Whitepaper: Application Logging — Security Best Practices

Author: Intesar Shannan Mohammed
Date: May 21, 2019

Abstract

This whitepaper aims to help organizations gain visibility into the internal and public-facing applications/API misuse, cyber threats, and data loss incidents, and more easily ensure customer compliance.

Audience

CISO, Security leadership, Engineering leadership.

Vulnerable Systems

Web Applications, APIs, SaaS, Cloud, IOT, Mobile Apps, SDKs, etc.

On average organizations are taking an additional 3-6 months[1] to report a breach incident. And it takes far more time to identify impacted users, data records and to complete the cleanup process. This trend is very evident from recent breaches at companies like Microsoft Outlook[2], Google Plus[3], Facebook[4], T-Mobile[5], etc. to name a few. It also means they're spending far more money on the audit and forensics.

The proposed strategy in this whitepaper is technology agnostic and will work across any architecture style, including APIs, Mobile/Web Applications backends, SDK's, etc.

Why WAF, Gateways, & Filters are not enough?

When it comes to real-time prevention, detection, and response, you can never be sure that your defense mechanisms are detecting all the known and the ever-increasing new threats.

For example, how do custom-built filters, WAF, or API Gateways perform in real-time? Relying on such defenses may block the most common attacks, including SQL, XSS, command injection, etc.
However, this approach doesn't protect your applications against the three most critical and most exploited vulnerabilities in today's world that deal with access-control flaws, privilege escalations, and sensitive data exposure.

Regardless of the safety mechanisms, you may have in place, you can never be sure that all existing and new threats are 100% detected and handled appropriately.

---

[1] "Most companies take over six months to detect data breaches | ZDNet." 19 May. 2015, https://www.zdnet.com/article/businesses-take-over-six-months-to-detect-data-breaches/. Accessed 16 May. 2019.
[2] "Microsoft admits Outlook.com hackers were able to access emails ...." 15 Apr. 2019, https://www.theverge.com/2019/4/15/18311112/microsoft-outlook-web-email-hack-response-comment. Accessed 16 May. 2019.
[3] "Google+ Exposed Data of 52.5 Million Users and Will Shut Down in ...." 10 Dec. 2018, https://www.wired.com/story/google-plus-bug-52-million-users-data-exposed/. Accessed 16 May. 2019.
[4] "Facebook data breach: Hundreds of millions of records exposed on ...." 4 Apr. 2019, https://www.cbsnews.com/news/millions-facebook-user-records-exposed-amazon-cloud-server/. Accessed 16 May. 2019.
[5] "T-Mobile was hit by a data breach affecting around 2 million customers ...." 24 Aug. 2018, https://www.theverge.com/2018/8/24/17776836/tmobile-hack-data-breach-personal-information-two-million-customers. Accessed 16 May. 2019.

Solution:

The solution is straightforward, and it requires you to track the user's access trails through your application logs. And, here are the 5 Best Security Practices to do that.

For every inbound/outbound request in your application, log the following few critical pieces of information.

1. *Logged-in-user*
2. *Logged-in-tenant*
3. *Action (Users find-all, etc.)*
4. *Query & path parameters (strings, dates, numbers, etc.)*
5. *Resource Id (Resources being created, read, updated, and deleted, etc.)*
6. *Important Resource properties (resource-name, username, ssn, city, etc.)*
7. *Application Errors (e.g. Resource-Not-Founds, Not-Entitled, 404, 500's, etc.)*
8. *Request body attributes (id, name, create-by, create-date, etc.)*
9. *IP Address*

Why log logged-in-user & logged-in-tenant information?

Logging the user, and tenant Id's will help you set the context for the incoming requests and also enable you to identify and track user activity.

Code Example:

```
public Response findAll(Integer page, Integer pageSize) {
  logger.info("Find all pageSize [{}] logged-in-user [{}] logged-in-tenant [{}]", pageSize,
loggedInUser, loggedInTenant);
 ...
}
```

Log output:

```
2019-05-14 18:35:53.903  INFO 54951 --- [nio-8080-exec-2]
c.f.fxt.rest.accounts.AccountController  : Find all pageSize [25] logged-in-user
[4028b881620688c001620689a3210000] logged-in-tenant
[4028b881620688c001620689a3210010]
```

Why log query and path parameters?

The query and path parameters are the most common subject of injection attacks e.g.

Code example:

```
public Response search(String keyword) {
  logger.info("Search by keyword [{}] logged-in-user [{}] logged-in-tenant [{}]", keyword,
SecurityUtil.getOrgId(), SecurityUtil.getCurrentAuditor());
  ...
}
```

Log output:

```
2019-05-14 18:39:45.655  INFO 54951 --- [nio-8080-exec-9]
c.f.fxt.rest.accounts.AccountController  : Search by keyword [' or 1=1] logged-in-user
[4028b881620688c001620689a3210000] logged-in-tenant
[4028b881620688c001620689a3210010]

2019-05-14 18:56:47.177  INFO 54951 --- [nio-8080-exec-2]
c.f.fxt.rest.accounts.AccountController  : Find all pageSize [10000] logged-in-user
[4028b881620688c001620689a3210000] logged-in-tenant
[4028b881620688c001620689a3210010]
```

The first log statement contains a malicious SQL injection script **“‘ or 1=1”** along with the logged-in-user. If you have some filters it can easily catch these issues early on and in addition, it will help you do better forensics later on as well. The technique will work across other injection categories, including command injection, XSS attacks, etc.

The second log statement captured a kind of **DDoS attack**, the user tried requesting a large number of resources i.e**. 10,000** items that can lead to a DDoS if done continuously for a lengthy period of time.

Why log Resource ID?

```
public Response update(String id, Account account) {
  logger.info("Update id [{}] logged-in-user [{}] logged-in-tenant [{}]", id, keyword,
SecurityUtil.getOrgId(), SecurityUtil.getCurrentAuditor());
  ...
```

}

Log output:

2019-05-14 18:41:47.829 INFO 54951 --- [nio-8080-exec-1]
c.f.fxt.rest.accounts.AccountController : Create **id** [**402881a1661eaa3d01661ecf618a0000**]
logged-in-user [**4028b881620688c001620689a3210000**] **logged-in-tenant**
[**4028b881620688c001620689a3210010**]

c.f.fxt.rest.accounts.AccountController : Read **id** [**402881a1661eaa3d01661ecf618a0000**]
logged-in-user [**4028b881620688c001620689a3210000**] **logged-in-tenant**
[**4028b881620688c001620689a3210010**]

2019-05-14 18:41:47.829 INFO 54951 --- [nio-8080-exec-1]
c.f.fxt.rest.accounts.AccountController : Update **id** [**402881a1661eaa3d01661ecf618a0000**]
logged-in-user [**4028b881620688c001620689a3210001**] **logged-in-tenant**
[**4028b881620688c001620689a3210011**]

The above log statements will allow you to build the threat detection map independent of the application and database in use and allow you to perform forensics. For example, you can extract information from the logs and create a map with 4 columns, resource, owner, tenant, is-authorized. Based on the resource owner and tenant you can easily determine if the incoming request was valid or a forgery.

Resource	Owner	Tenant	Is-Authorized
R1 (Create)	**A1**	**T1**	Authorized Access
R1 (Read)	A2	**T1**	Authorized (A2 is part of the tenant T1)
R1 (Update) [Threat]	**B1**	**T2**	Unauthorized (B1 is trying to access data from T1 tenant)

Why Important Resource Properties?

Logging Important resource properties are critical to prevent stored command injections (SQL, XSS, or Linux, etc.). If these properties are not properly validated then malicious code can be stored in the database using these properties.

There are a couple of ways these malicious codes can be later exploited either through internal SQL or through version control frameworks like Flyway, Liquibase, dbMigration .Net, etc.

Here is an example of how stored code can be exploited from Oracle reference: https://download.oracle.com/oll/tutorials/SQLInjection/html/lesson1/les01_tm_attacks2.htm.

1. Suppose you have a Web-based application which stores usernames alongside other session information. Given a session identifier such as a cookie you want to retrieve the
current username and then use it in turn to retrieve some user information. You might therefore have code for an "Update User Profile" screen somewhat similar to the following:

   ```
   execute immediate 'SELECT username FROM sessiontable WHERE session
   ='''||sessionid||'''' into username;

   execute immediate 'SELECT ssn FROM users WHERE
   username='''||username||'''' into ssn;
   ```

   This will be injectable if the attacker had earlier on the "Create Account" screen created a username such as:
   `XXX' OR username='JANE`

   Which creates the query:
   `SELECT ssn FROM users WHERE username='XXX' OR username='JANE'`

   If the user `XXX` does not exist, the attacker has successfully retrieved Jane's social security number.

2. The attacker can create malicious database objects such as a function called as part of an API, or a maliciously named table by using double quotation marks to introduce dangerous constructs.

   For example, an attacker can create a table using a table name such as `"tab') or 1=1--"`, which can be exploited later in a second order SQL injection attack.

Why log exceptions and errors?

Exceptions are the most critical piece of evidence or a sign of an early probing mission. Most exceptions or errors occur as a result of some defense in the application logic against malicious data or access to unauthorized data. These errors should be treated with the highest importance and should be logged with the logged-in-user id. You can easily catch all the exceptions in your application using AOP or filter mechanisms.

2019-05-14 19:04:17.328 INFO 54951 --- [nio-8080-exec-4]
c.f.fxt.rest.accounts.AccountController : Find by id [**402881a16aa87e13016aa87f45ff0000**] **logged-in-user** [4028b881620688c001620689a3210000] **logged-in-tenant** [**4028b881620688c001620689a3210010**]
2019-05-14 19:04:17.352 WARN 54951 --- [nio-8080-exec-4]
c.f.fxt.rest.base.ExceptionTranslator : Index: 0, Size: 0
java.lang.**IndexOutOfBoundsException**: Index: 0, Size: 0

This log trail is an example of a **phishing attack**. Because the Id doesn't exist in the database.

Conclusion

1. Logs provide a single source of truth for all kinds of threats.
2. Allows you to capture data on existing and as well as new threats.
3. Much easier to do round-the-clock monitoring.
4. Allows you to learn and reinforce your defense mechanism against missed threats.
5. Allows you to respond and block accounts and tenants under attack.

ROI:

The ROI is several times compare to your one-time investment in the strategy. For example, if you have an API with 100 endpoints, this strategy will require you to add just 100 log statements and that can take no more than a day or two's effort.

Reviewers:

- Aaron Alvarez (FX Labs)
- Riyaz Shaik (FX Labs)
- Luqman Shareef (FX Labs)
- Maqbool Khan (Google)
- Amjad Afanah (FX Labs)

References:

https://www.giac.org/paper/gsec/501/information-security-process-prevention-detection-response/101197