Building Security In
Editors: John Steven, jsteven@cigital.com
Gunnar Peterson, gunnar@arctecgroup.net

How to Do Application Logging Right

As threats shift toward applications and as more companies struggle with compliance mandates, the need for useful, comprehensive application logging can only increase. Here we provide guidance on application logging to application developers and architects and to security professionals.

ANTON CHUVAKIN
SecurityWarrior Consulting

GUNNAR PETERSON
Arctec Group

Application Logging Today

Organizations have finally gotten network device logging and—to some extent—server logging under control. However, after getting used to neat Cisco Adaptive Security Appliance or other firewall logs and Linux "password accepted" messages, security incident investigators trying to respond to the next wave of attacks have been thrust into the horrific world of application logging.

Problems with many application logs are truly staggering. Logs are often missing, they omit critical details, or they have no standard form or content. On top of this, many security practitioners must deal with debugging logs masquerading as security audit logs. Table 1 illustrates the key differences between the two.

Debugging logs appear in application frameworks more frequently than well-designed security audit logs. However, using them for investigations is often an exercise in frustration because they might not contain key details needed for incident response and forensics.

So, sometimes we must deal with log messages like these:

- `Aug 11 09:11:19 xx null pif ? exit! 0` (which carries absolutely no meaning),
- `Message 202 User transitioning priv level` (which conveniently omits the actual user identity—the use of "secret" numeric codes also causes trouble if no documentation is available), or
- `userenv[error] 1040 XYZ-CORP\wsupx No description available` (which scores points for both honesty and uselessness).

In light of this, how do we guide application developers and architects toward creating security audit logs that are useful for forensics and monitoring and that help satisfy compliance mandates such as the Payment Card Industry Data Security Standard (PCI DSS)? We can start by establishing criteria for good security audit logs (which we just call "logs" from now on).

Logging Criteria

From a high level, the best logs tell you exactly what happened, and when, where, and how. Such logs are suitable for manual, semiautomated, and automated analysis. Ideally, you can analyze them without having the application that produced them—and definitely without having the application developer on call. From the log management viewpoint, the logs can be centralized for analysis and retention. Finally, they won't slow the system down and can be proven reliable, if used as forensic evidence.

What to Log

So, what types of events should you log?

The first type is *authentication, authorization, and access* events:

- successful and failed authentication or authorization decisions;
- system access, data access, and application component access; and
- remote access, including from one application component to another in a distributed environment.

The second type is *changes*:

- system or application changes (especially privilege changes),
- data changes (including creation and destruction), and
- application and component installation and changes.

The third type is *availability issues*:

- startups and shutdowns of systems, applications, and application modules or components;
- faults and errors, especially errors affecting the application's availability; and

Table 1. Comparing security audit logs and debugging logs.

	Security audit logs	Debugging logs
Intended consumers	Security and audit personnel	System operators and developers
When the logger is on	Always	Sometimes
Message content	Attacks, activities, and faults	Faults, failures, and errors
Scope of what to log	Known in advance	Unknown
Length of usefulness	Years	Hours or days

- backup successes and failures that affect availability.

The fourth type is *resource issues*:

- exhausted resources, exceeded capacities, and so on;
- connectivity issues and problems; and
- reached limits.

The final type is "*badness*" or *threats*:

- invalid inputs and other likely application abuses and
- other security issues known to affect the application.

Creating a comprehensive "what to log" list for every application and organization is impossible. However, our list should provide a useful starting point for your custom applications, especially those dealing with regulated data such as payment cards or sensitive personal information.

What to Include

Next, what data should you log for each event, and at what level of detail should you log it? The philosophy of relevant log details goes back to ancient Greece (http://en.wikipedia.org/wiki/Five_Ws) and focuses on the "Six Ws":

- Who was involved?
- What happened?
- Where did it happen?
- When did it happen?
- Why did it happen?
- How did it happen?

(No, we can't explain why the sixth W is actually an H.) This ancient wisdom applies perfectly to logs and helps define a useful, unambiguous log entry.

On the basis of the six Ws, the following list provides a starting point for what to include:

- The *username* helps answer "who" for those events relevant to user or administrator activities. In addition, it's helpful to include the name of the identity provider or security realm that vouched for the username, if that information is available.
- The *object* helps answer "what" by indicating the affected system component or other object (such as a user account, data resource, or file).
- The *status* also helps answer "what" by explaining whether the action aimed at the object succeeded or failed. (Other types of status are possible, such as "deferred.")
- The *system*, *application*, or *component* help answer "where" and must provide relevant application context, such as the initiator and target systems, applications, or components.
- The *source* helps answer "from where" for messages related to network connectivity or distributed application operation.
- The *time stamp* and *time zone* help answer "when." The time zone is essential for distributed applications. In addition to the time stamp and time zone, some high-volume systems use a transaction ID.
- The *reason* helps answer "why," so that log analysis doesn't require much digging for a hidden reason. Remember, the log's customers are the security and audit personnel.
- The *action* helps answer "how" by providing the nature of the event.
- In addition, the *priority* helps indicate the event's importance. However, a uniform scale for rating events by importance is impossible because different organizations will have different priorities. (For example, different companies might have different policies regarding information availability versus confidentiality.)

So, a useful log message might look like this:

```
2010/12/31 10:00:01AM
GMT+7 priority=3,
system=mainserver,
module=authentication,
source=127.0.0.1,
user=anton(idp:solar),
action=login,
object=database,
status=failed,
reason="password
incorrect"
```

This message has a field explaining the failure's reason. Also, it isn't in XML; human readability is useful in logs, and computers can deal with `name=value` pairs just as well as with XML.

What Not to Include

Certain details should never be logged. Some examples are obvious: logs should never contain application or system passwords.

Table 2. Defending against SQL injection attacks.

Layer	Injection defense	What the logger will report
Presentation layer	Form input validation	An invalid-input event—that is, a failed white list or blacklist input validation event
Business logic layer	Business logic, rules, or policies	A failed authentication, authorization, or access event—that is, failed validation based on business logic, rules, or policies
Data access layer	Prepared statement or parameterized query	A resource issue event—that is, an SQL syntax error event

(Sadly, this sometimes still happens with Web applications.)

Centralization

As we mentioned before, easy centralization of logs is essential for distributed log analysis across either multiple systems or multiple application components of a distributed application. Although syslog has been a flawed but de facto standard of log centralization owing to its easy User Datagram Protocol delivery, modern cross-platform application frameworks call for the publish/subscribe model for log delivery. In this case, a security-monitoring tool can request a subscription to a particular type of logged event—and receive all relevant logs in nearly real time, if needed.

Know Your Customer

When you add logging functionality to your application, you're not just building security in, you're building visibility in.[1] That visibility is the information in the log that will be viewed by your log's customer—the incident responder. To maximize the log's usefulness, you must understand the incident responder's requirements. These might be a particular audit record format, integration with network security monitors, or listening for specific types of events.

One thing is for sure: application developers can gather context that's simply unavailable anywhere else in the system. The application is the concrete implementation of the software's structure and behaviors from the business logic, business rules, enterprise policies, and Web front ends to the data structures and storage. This gives the application context, which is everything to an incident responder.

Locating the Logging Service

The logging service's location relates directly to the type of events the logger can see and the data and context available to the logger. The logger is responsible for discerning the event type, gathering context, and writing information. To discern events, the logger must be able to view the event's source, its payload, and the object. Additional context includes such information as the authority and security domains.

Given a Web application attack such as SQL injection, a logger in the presentation layer might be unable to discriminate valid input from invalid input (an attack). But in the business logic layer (where business rules are applied) or data access layer (where SQL statements and data connections are made), increased knowledge of the object environment provides better visibility for the application as a whole, and specifically the logger, to flag input as a possible attack.

Table 2 describes how a typical three-layer architecture defends against SQL injection. It also gives an example of how the visibility at each layer drives what the logger reports.

The same malicious user input might be reported differently at each layer. To help correlate these reports, the logger often adds a transaction or message exchange ID to facilitate log analysis when reconstituting the events.

Sanitizing Audit Records

Typically, logging subsystems are placed to detect events around sensitive assets, which means that they'll come into contact with sensitive data. Sanitizers can filter and remove sensitive data from logs. A sanitizer's location (see Figure 1) is important because it determines whether sensitive data is filtered by the log browser or removed from persistent storage (sensitive data is never stored in the log).

Storage Forecast—Cloudy with a Chance of Compromise

Most systems store logs inside the enterprise, but as with many IT areas, the cloud offers new opportunities and potential solutions and problems. The cloud has proven to be an effective way to store data. However, because in the cloud storage model, the data is stored and possibly processed outside enterprise security, challenges remain owing to requirements for encryption and other controls. For example, PCI DSS provides a starting point for log storage requirements, but meeting them might be difficult in a cloud model.

In addition to our basic conclusion—You must log!—we

must remind you that logging's importance will only grow. In particular, the need to analyze application behavior for security issues across distributed and, soon, cloud-based applications calls for us to finally get logging under control.

Software architects and developers must "get" logging; there's no other way. This is because infrastructure logging from network devices and operating systems won't cut it for detecting and investigating application-level threats. Security teams will need to guide developers and architects through useful, effective logging.

Certainly, logging standards such as Mitre Common Event Expression (cee.mitre.org) will help, but several years might pass before they develop and their adoption increases. Pending a global standard, organizations should quickly build and implement their own standard using the guidelines we presented. They should also use standard-language APIs, libraries, and logging mechanisms while ensuring that their logs record all relevant information. □

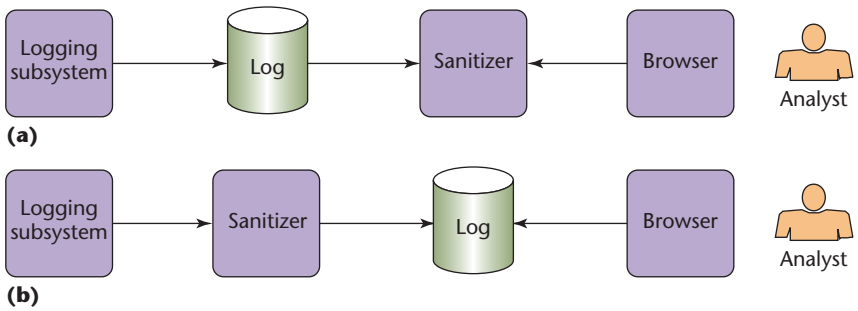

Figure 1. The sanitizer's location is important. It determines whether sensitive data is (a) filtered by the log browser or (b) removed from persistent storage.

Acknowledgments

We thank Raffy Marty of Loggly for his thoughtful review of the draft article.

Reference

1. R. Bejtlich, "Build Visibility In," blog, 13 Aug. 2009; http://taosecurity.blogspot.com/2009/08/build-visibility-in.html.

Anton Chuvakin *is a security consultant specializing in log management and Payment Card Industry Data Security Standard compliance. Contact him at anton@chuvakin.org; www.chuvakin.org.*

Gunnar Peterson *is managing principal of Arctec Group. Contact him at gunnar@arctecgroup.net.*