Hidden Implementation

(Mini-Pattern)

Abstract

The *Hidden Implementation* pattern limits an attacker's ability to discern the internal workings of an application—information that might later be used to compromise the application. It does not replace other defenses, but it supplements them by making an attacker's job more difficult.

Problem

Before an attacker can wage a successful attack on a system, he or she must understand how the system operates. Current systems tend to provide the attacker with a great deal of information about both the standard components and the custom elements. As a result, it can be very easy for a potential attacker to assess whether an attack is likely to be successful—or at least go undetected—before having to do anything that might expose him or her to risk.

There are numerous specific occasions in which implementation details are revealed, including the following:

- Most Web servers greet the client with a detailed description of the Web server software, even going so far as to indicate the precise patch level.
- Many Web application systems use telltale file extensions (.asp, .jsp, .cfm) that indicate what sort of dynamic content generation is taking place.
- Web authoring environments often default to a standard directory layout that reveals the specific product used for generation.
- Application error messages can give detailed information about the software that failed and the nature of the failure.
- Field names in Web forms and hidden variables leak application variable names.

A savvy attacker can use all of this information to increase the effectiveness of an attack.

Solution

The *Hidden Implementation* pattern forces examination of all communication with the client for anything that might provide information about the internal workings of the system. If the system has already been designed, then consider implementing changes or installing filters that sanitize the data received by the client. Otherwise, the system should be designed in such a fashion that would make it impossible or difficult for an attacker to learn how the system is implemented.

One must examine the system to determine where information might be gathered and the potential impact this could have on the system.

- Understand what pieces of information are sensitive; consider how it could be modified or abused.
- Do not expose anything that is not absolutely necessary.
- Go back and look at the system from an attacker's point of view...what could the attacker learn? (See *Red Team the Design*.)
- Examine the information the user sees on several levels -- what does it reveal about the technology used to implement the system, the location of files, the content of the system data, etc., and consider how feasible it is to hide each of these.

All errors should be mapped to generic "unavailable" screens and the specific details logged to an internal debugging log.

Remove debugging capabilities, and comments in generated code.

Hidden Implementation presents some challenges:

- It is difficult to imagine all the negative uses of information.
- It may be difficult to hide all the important aspects of an application.
- The platform or tools used to build the system may not allow you to hide some implementation clues.
- It may require significantly more design and development time hide pertinent information.

Related Patterns

- *Account Lockout* – a related pattern; authentication failure screens can be made to reveal no information about the security policy. For example, a failed login screen can give the same generic message, regardless of whether the user ID was incorrect, the password was incorrect, or the maximum number of failed attempts was exceeded.
- *Authenticated Session* – a related pattern; sessions can be used to prevent leaking important information to clients, instead of telling the client what its username is you can instead assign the client a session identifier that maps to the information which is stored locally.
- *Minefield* – a related pattern; the application of implementation hiding principles cause the potential hacker to work much harder to understand the workings of the system. By making the reconnaissance more difficult and time-consuming, the attacker is more likely to be set off the well-tuned or customized intrusion detection system of a minefield.

- *Red Team the Design* – a related pattern in which red teams can examine the system from an attacker's point of view to look for parts of the design that reveal implementation details.

References

None.