Application Layer Encryption: Protecting against Application Logic and Session Theft Attacks

Whitepaper

The security industry has extensively focused on protecting against malicious injection attacks like SQL injection and Cross-site Scripting. However, application logic attacks are equally dangerous, yet stealthier. Most intrusion detection systems, including first-generation Web Application Firewalls (WAFs) cannot detect them. This paper discusses these attack vectors and the techniques to protect against them.

Application Logic attacks are stealthier than traditional SQL and XSS Injection attacks. They can go unnoticed by all security layers such as code or vulnerability scanning, blocklist and whitelist filtering, etc., and evade detection even in audit trails and forensics.

Attacks Targeting Application Logic

In the early days, IDS and IPS products started including exploit signatures for input validation against injection attacks like SQL and XSS Injections. However, these were easily bypassed using attack obfuscation techniques. To counter this, WAFs started offering de-obfuscation mechanisms that normalize inputs, followed by regex style pattern matching to be more flexible in detecting attacks. Zero-day attacks still remained a concern, and to combat them, the use of positive security became prevalent in WAFs, which attempt to validate inputs against a whitelist, rather than a blocklist. To ease the use of the positive security model, automated policy-learning mechanisms were introduced.

While such automated learning and anomaly detection mechanisms are a great step forward, especially for securing against SQL and XSS injections, they can still be blind to application logic attacks. Simply stated, it is just not possible to have blocklists or whitelists to protect against attacks in the application's logic itself.

Application logic attacks attempt to manipulate application inputs in order to directly target the application logic, rather than the peripheral environment – databases, operating systems, and development frameworks on the server side, and JavaScript Interpreters on the client side. The typical surface of such attacks is the integrity and the logic validation mechanism of the application itself, e.g. business logic, workflow, account creation, authorizations, logins, account modifications, shopping carts, session management, gift card or promotional code usage, fund transfers, etc.

Such an attack surface is exposed through URL paths, parameters, hidden parameters, session IDs, cookies, and headers. Attack input blends into the normal usage patterns and does not trigger negative or positive security policies. Besides, application-scanning technologies like code scanning (SAST) or vulnerability scanning (DAST) fail to detect them. More alarmingly, audit tools and procedures can be completely blindsided too.

Logic attacks frequently target data that passes back and forth between server and clients. Such data is exposed via URLs, cookies, hidden parameters, referrers, etc., and can be directly manipulated by the attackers.

Attacks targeting the Application Logic

To illustrate application logic attacks, below are several sample attacks adopted from OWASP documentation that target the web application's integrity and logic validation mechanisms.

Example 1

A medical record on a system one might have a URL such as:

http://example.com/RecordView?id=12345

If the application does not check that the authenticated user ID has read rights, then it could display data to the user that the user should not see.

Example 2

When a web application uses hidden fields to store status information, a malicious user can tamper with the values stored on his browser and change the referred information. For example, an e-commerce shopping site uses hidden fields to refer to its items, as follows:

<input type="hidden" id="1008" name="cost" value="70.00">

In this example, an attacker can modify the "value" information of a specific item, thus lowering its cost.

Example 3

An attacker can tamper with URL parameters directly. For example, consider a web application that permits a user to select his profile from a combo box and debit the account:

http://www.attackbank.com/default.asp?profile=741&debit=1000

In this case, an attacker could tamper with the URL, using other values for profile and debit.

Other parameters can be changed including attribute parameters. In the following example, it's possible to tamper with the status variable and delete a page from the server:

http://www.attackbank.com/savepage.asp?nr=147&status=read

Modifying the status variable from "read" to "del" could delete the page.

Below, we present some key techniques used by the Barracuda Web Application Firewall to provide protection from such attacks.

Reducing the Attack Surface #1: Encrypting URL Data

Manipulating URL path or parameters is often the first and easiest step in attacking application logic. In examples 1 and 3 above, the URL parameters "id", "profile", and "debit" are targeted.

The fact that these parameters which are plugged into the application logic are directly exposed in the URL is what makes these attacks possible. Encrypting and tamper-proofing them would completely deny these attacks. The Barracuda Web Application Firewall achieves exactly this by using URL Encryption.

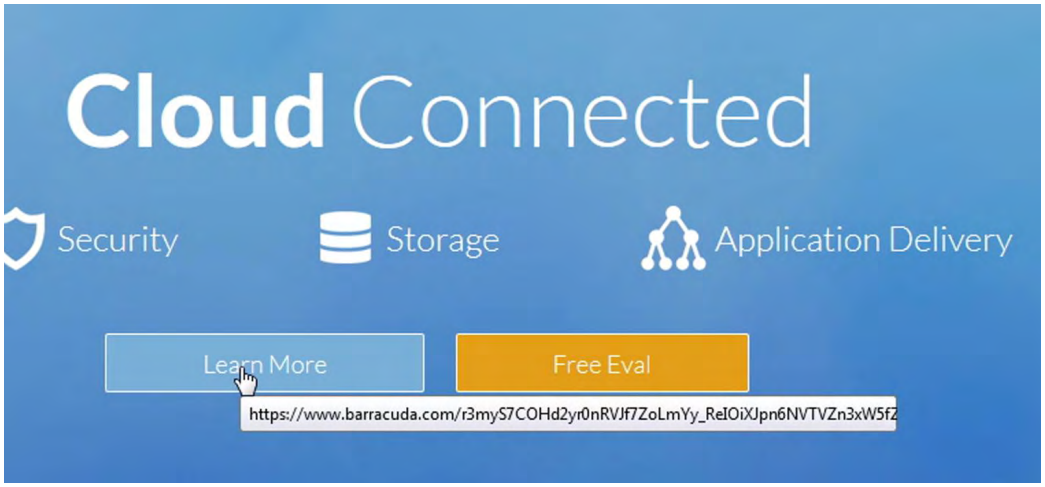
When using URL Encryption, the Barracuda Web Application Firewall encrypts all the URLs that are visible to users. It ensures that URLs are encrypted on the way out and decrypted on the way back in. Users never get to see the actual parameters or paths of the URL.

The nature of the encryption/decryption process ensures that decryption will fail even if a single character is tampered with. Since this process is completely transparent to the protected applications, no change is required on them.

On encryption, the URL that is sent out to the users resembles the following:

http://example.com/9gh7filg6320ixtqa2bc0r

Notice that all the vulnerable parameters above are unavailable to attackers. In addition to logic attacks prevention, this also guarantees foolproof protection against zero-day attacks. The following is a screenshot that shows URL Encryption in action, notice the mouseover "Learn More" shows a completely encrypted URL:



Reducing the Attack Surface #2: Tamper-Proofing Hidden Parameters

Another attack surface that attackers frequently target for logical vulnerabilities are hidden parameters.

Hidden parameters are fields used by developers to embed state information within forms. Browsers do not display them and the assumption is that this information will remain unchanged when the client browser submits the form. However, attackers can easily view and manipulate them through commonly available tools such as Paros and Webscarab proxy.

Example #2 above is an example of such an attack.

The Barracuda Web Application Firewall prevents hidden parameter tampering by embedding encrypted hashes of the hidden parameters and comparing them with recomputed hashes when the form is submitted. This ensures that the data is not tampered with either on the client end or in transit.

Reducing the Attack Surface #3: Protecting Session Tokens in HTTP Cookies

To maintain user sessions, stateful web applications store SessionID tokens in HTTP cookies. Stealing the SessionID is as good as stealing the username and password of the user. Cookie theft could be either due to a man-in-the-middle/browser (MITM/MITB) attack or an attacker guessing SessionID due to insufficient entropy in the SessionID generating process.

As an example, in 2013, a security researcher exposed how he hacked Facebook Employees Secure Files Transfer service (http://files.fb.com). The crux of the attack was a parameter called "referer" in the cookie, which is base 64 encoded. It actually contains the email address of the user. Changing the email to that of another user and re-encoding it in base 64 allows you to change the victim's password. A very old and simple trick from the hacking handbook, affecting one of the largest and most modern of web sites.

The Barracuda Web Application Firewall provides two strong protection techniques for securing against such attacks. One, it encrypts all cookies which makes them impossible to predict, as well as blocks the request if any tampering is found. Second, it provides cookie replay protection, which associates a cookie with the user's IP address. If the cookie is seen from a different IP, the request is blocked.

Cryptographic protection of application tokens is the only way to trust the integrity of the data sent back and forth between the web application and the browsers. This is true even over SSL, as it provides protection only in transit, but not on the endpoints.

Reducing the Attack Surface #4: Protecting Session Tokens in URLs

Often, in cookieless sessions, the SessionID tokens are not stored in cookies but are embedded in the URLs themselves. For example:

http://www.example.com/(S(pyt3py66t21z5v55vlm25t7y))/orderform.aspx

Such SessionID tokens can easily leak when they are stored by servers, proxies, and user agents, and shared by users, e.g., via copy/pasting or screenshots. The referrer header can also leak the sessionID to external web sites.

As described above, the URL Encryption process in the Barracuda Web Application Firewall completely encrypts the URL, hence this attack surface is mitigated.

Reducing the Attack Surface #5: Foolproof Referrer Enforcement

In a multi-step workflow, users are not expected to be able to skip intermediate steps, for example those that are related to identity verification. If a user can predict the URL in, say, step 5, they can "jump" directly from step 2 to step 5, skipping application safeguards that may have been affected in steps 3 and 4.

To overcome this, many applications implement referrer enforcement, where a step like step 5 would ensure that the client browser has been referred to it from step 4. However, faking the referrer is a trivial job, so this control can be easily bypassed.

However, if you were to use URL Encryption across the workflow, then this attack vector is nullified, as the attacker has no way of predicting the step 5's referrer header without going through step 4. The referrer header, in this case, would be the dynamically-generated, encrypted URL for step 4.

Conclusion

Most of the attacks described above are critical, however traditional vulnerability scanning or request filtering approaches would not detect them. The application can simply not trust input data from users. Application layer encryption is the only reliable mechanism to guarantee against malicious tampering of such data. This is not the same as using HTTPS, since HTTPS only provides in-transit protection. The Barracuda Web Application Firewall implements a variety of techniques to secure against such application logic and session theft vulnerabilities.

In a legacy heterogeneous environment using multiple development frameworks, embedding application layer encryption within the source code is a non-starter. The Barracuda Web Application Firewall can achieve this on-the-fly without a single change on the web servers.

About Barracuda Networks, Inc.

Barracuda provides cloud-connected security and storage solutions that simplify IT. These powerful, easy-to-use, and affordable solutions are trusted by more than 150,000 organizations worldwide and are delivered in appliance, virtual appliance, cloud, and hybrid deployments. Barracuda's customer-centric business model focuses on delivering high-value, subscription-based IT solutions that provide end-to-end network and data security. For additional information, please visit barracuda.com.

Barracuda

Barracuda Networks Inc.
3175 S. Winchester Boulevard
Campbell, CA 95008
United States

1-408-342-5400
1-888-268-4772 (US & Canada)
info@barracuda.com
barracuda.com