ZTEdge™
By Ericom Software

WAI and the OWASP Top 10

Introduction

Security is a key strategic requirement for any modern business. With business activities increasingly shifting to web apps, securing their operations is essential. Web Application Firewalls (WAFs), which were originally designed and architected in the early 90's to solve for a different set of application access scenarios than we have today, fail to address the issues as needed to secure application operations in the wilds of the internet.

Web Application Isolation

Web application isolation (WAI) takes a fundamentally different approach to protecting applications and the data within them than traditional WAF solutions.

WAI inverts remote browser isolation to airgap networks and apps from malware on user devices, and applies granular user-level policies to control which applications each user can access, how, and which actions are permitted for each user, in each app. SaaS and web application access may be restricted to specific IP addresses.

Some WAI solutions require dedicated software or agents to be installed on every device. Cloud-based solutions that require no endpoint agents are ideal, not only because they are more convenient but also because web app security is most essential for workers whose devices are unmanaged, like 3rd party contractors.

The OWASP Top 10 Security Risks to Web Applications

To help security leaders and operators understand the role that WAI can play in securing their environments, this paper maps WAI controls against the OWASP Top 10 Web Application Security Risks, the globally recognized framework for web application security.

As demonstrated below, WAI's isolation-based approach of air-gapping applications from the risks of unmanaged devices (e.g. devices used by 3rd parties and contractors) and the broader internet is indisputably superior to the traditional "hopefully detect then try to defend" WAF approach.

Best Practice	WAI Approach
Broken Access Control	
The best defense against this vulnerability is to deny access by default – a non-starter for modern enterprises. Users need access and blocking everything by default is not a feasible approach. Instead, access controls must be implemented and used across the application. Domain models need to enforce business limit requirements, and web server directory listings must be disabled. This prevents file metadata and backup files from being present inside of web roots and helps limit reconnaissance of critical information. Logging issues such as access control failures and detection of repeated failures is important, as they can lead to proactive controls being implemented before a warning becomes a security event. Automated attack tools are limited by instituting rate limits against APIs and controller access. Finally, JWT tokens need to be invalidated on the server when a logout occurs.	An enforcement mechanism can deny access to web application objects and pages based on crafted back-end policy configuration from the WAI controller. Upon successful authentication, the WAI controller policy enforces explicit rights based on users' associated roles/groups. WAI obfuscates the web application URL and page source so attackers cannot access local files. Unauthenticated users cannot access the web application anonymously.
Cryptographic Failures	
Strong cryptographic controls and protocols should always be used by applications. Encryption must be employed both in transit and at rest and data that is stored or accessed should have applied cryptographic controls.	WAI's TLS 1.3 overlays any unsecured data transmission. All application data is obfuscated using pixel-based rendering. Sensitive data stored and processed by the web application is air-gapped from the internet and is not cached on the user's endpoint. Blocking file download can be applied as an additional control.
Injection	
Only valid requests or posts should be allowed, and controls should be applied to manage only valid requests. Applications should limit inputs to defend against injection commands and filters should be in place to validate a connection or application query.	Injection attacks can occur in normal user input forms as well as in hidden web fields. When using pixel modes there are no FORM and fields at the client, so it is impossible to inject to fields. URL injection is also impossible due to obfuscation of the app URLs. It is also advisable to monitor outbound responses returned to the user to detect information leakage resulting from a successful injection attack.

Best Practice	WAI Approach
Insecure Design	
Creating a repeatable hardening process that is fast and easy to deploy is essential when deploying a new environment but is also hard to do, due to the speed of development and sprawl of application infrastructure. The development deployment process needs to be automated to allow for effortless deployments. The platforms that are used must be equipped with only the features that are necessary, as this minimizes the attack surface that can be leveraged by an attacker. Administrators need to stay up to date with all patch releases and security notes, as well as updates. Segmented application architecture is one of the strongest defenses against security misconfiguration. Finally, automated processes should be implemented that verify how effectively the configurations of each environment have been implemented.	WAI adds secure access without the friction of additional software development and infrastructure redesign. WAI does this by: • Isolating the web code, making it 'invisible' to the internet and eliminating the attack surface. • Adding multi-factor authentication to minimize unauthorized access. • Using an isolation layer that provides an airgap to segment web servers from brute force attacks originating from the internet.
Security Misconfiguration	
Infrastructure components must be configured correctly for operations and should be regularly checked and updated for weaknesses and vulnerabilities.	To properly secure a web application, not only the software must be properly locked down, but all other integral infrastructure components must be locked down as well. It is also important to conduct regular and thorough audits to ensure that controls have been implemented properly and remain firmly in place. Continuous vulnerability scanning is the key to ensuring nothing is missed. However, if a vulnerability is discovered, WAI can provide immediate protection rather than waiting on a time-intensive fix. Also, because any modification to code—whether to resolve a vulnerability or fix a defect—can introduce additional vulnerabilities, WAI becomes a critical component for ensuring uptime with minimal risk, so developers have the necessary time to produce a stable and secure solution.
Vulnerable and Outdated Components	
Regular upgrade cycles and patch management must be in place and should be automated to work at scale across hybrid infrastructure.	The key to a secure environment is to keep components updated wherever you can. Where you can't, compensating controls such as strong WAI can be used to block exploitation of known vulnerabilities, while keeping software intact and operational. WAI can buy time while patches are developed and rolled out—and it also protects against any subsequent attack variants that are common when exploit code hits the web.
Identification and Authentication Failures	
Multi-factor authentication is one of the most effective methods of preventing unauthorized access via broken authentication exploits. Product owners can also minimize their exposure by ensuring that their applications don't ship with default credentials and passwords. It's also important to protect user choices for passwords by enforcing minimum password requirements such as length, complexity and password reuse restrictions and rotation. Other steps include username and password recovery, as well as the use of the same message for all outcomes by hardening the API pathways against account enumeration attacks. All login failure events need to be logged, and if credential stuffing or brute-force attempts are detected, then system administrators need to be alerted. Server-side authentication is important because it prevents attackers from breaking authentication on a local system. Session IDs must not be displayed in the browser, as this allows attackers to use session keys as an attack vector.	WAI secures user identification and authentication by: Including a built-in identity provider with multi-factor auth support. Mandating the use of strong (complex) passwords to meet compliance with best practices. Brute-force detection by permanently or temporarily disabling a user account if the number of login failures exceeds a specified threshold. The WAI authentication layer secures authentication-related vulnerabilities where: • Default passwords are being used. • Users are sharing accounts and passwords. • The application cannot support multi-factor authentication by itself. WAI also supports external SAML 2.0, Open ID Connect, and social identity providers such as Azure Active Directory, Okta, Ping Identity, Google login, etc.

ZTEdge™
By Ericom Software

Best Practice	WAI Approach
Software and Data Integrity Failures	
Data must be classified to be best protected. This approach helps the team know what data has been processed, stored, or transmitted by the application or systems that are running, and to understand the value of that data for the business. Another potential solution to this problem is to not store any sensitive data in the first place. Data can be tokenized and truncated. All data that is at rest must be encrypted. All data in transit must be encrypted with security protocols such as TLS along with Perfect Forward Secrecy ciphers (PFS). Disabling the caching of sensitive data and only using salted hashes is also advisable.	WAI's authentication layer can help by making sure that usernames cannot be validated from server response codes: an incorrect username error and incorrect password should generate the same (or generic) error message. WAI policy can force the browser to employ security controls related to access to and transactions with sensitive data, for example, disabling clipboard copy/paste. WAI uses TLS 1.3 to secure data in transit. All application data is obfuscated using pixel-based rendering. Sensitive data is never stored on the user's endpoint. Blocking file downloads can also be applied to prevent leakage of sensitive data.
Security Logging and Monitoring Failures	
Everything should be logged and leveraged for additional insight and understanding of what is taking place for applications that are in use.	WAI allows standardization of access logging for web applications based on policy controls—and logging of that information off-box (SIEM) for further analysis and reporting.
Server-Side Request Forgery	
Whitelist Domains in DNS is a best practice to stop SSRF. This happens by whitelisting any domain or address that the application accesses. By also not allowing the server to send raw responses this threat can be managed as well. An application should never send a raw response body from the server to the client. Responses that the client receives need to be expected. Lastly, by enforcing correct URL schemas SSRF can be stopped. Examples include: Allow only URL schemas that an application must use to access backend and front-end resources. There is no need to have ftp://, file:/// or even http:// enabled if you only use https://. Applied isolation and segmentation and ZTNA-focused access can also be tied into this fix for SSRF. Authentication of all services should be enabled on any service that is running inside a network even if the application doesn't require that service to operate. Services such as memcached, redis, mongo, and others don't require authentication for normal operations, but this means they can be exploited and should be made to enable authentication. Sanitizing and validating inputs is necessary as well. By enforcing only parsed and sanitized user input sent to an application SSRF is managed. After sanitization validation should be employed to ensure that nothing malicious was allowed to pass through the application via that user input.	WAI hides internal servers from the internet so they cannot be port scanned. WAI obfuscates the web application URL so attackers cannot access local files and internal services by manipulating the application's URL. WAI acts as an airgap to the internal application and its dependent services - defending it against denial of service attacks.

Conclusion

ZTEdge Web Application Isolation airgaps your applications from web-based threats and the risks of unmanaged devices, providing an application security solution that is designed for the modern age: An age in which the productivity of employees, contractors, and partners depends on having simple, secure access to applications in the cloud (private apps or SaaS) and on-premises, using a wide array of devices.

Please contact Ericom Software to learn more about WAI.