Creating Attack-Aware Software Applications with Real-Time Defenses

Colin Watson, OWASP
Michael Coates, OWASP
John Melton, OWASP
Dennis Groves, OWASP

Abstract. Attack-aware software applications provide attack detection and real-time defensive response with a very low false-positive rate. This technique allows an application to detect and neutralize a threat before the attacker exploits a known or unknown vulnerability. The approach is especially suited to software applications with high information assurance requirements such as in the defense, critical national infrastructure, and financial service sectors to protect against cyber espionage, fraud, business logic abuse, tampering, and theft. The Open Web Application Security Project (OWASP) has developed a methodology, documentation, code and pilot demonstration which can be freely used to apply the concepts; this project is called AppSensor.

Introduction

Information systems and data are being targeted relentlessly by skilled and motivated adversaries who are well resourced and have excellent tools. These attackers, who may be backed by organized crime, governments, or commercial enterprises, identify and exploit vulnerabilities in applications themselves to access sensitive and secret data.

In this article we discuss why conventional application defenses are not a solution to these types of advanced attacks, and explain an initiative from OWASP [1] that utilizes self-defense to provide real-time detection and response.

Conventional Defensive Measures

A fundamental starting point for secure applications is during the development lifecycle. The goal is robust code—designed, developed, configured and operated on hardened operating systems. However, this does not mean the software is impregnable; unknown vulnerabilities almost certainly exist, new vulnerabilities are introduced with software code changes and attackers examine and probe applications to find them. Once they identify one or more vulnerabilities, they discover ways to exploit them. It is very hard to detect these custom attacks using conventional defensive measures.

Some techniques often thought to defend applications include using Transport Layer Security (TLS), network firewalls, application gateways (e.g., web application firewalls, XML appliances, cross-domain guards), and network/host Intrusion Detection and Prevention Systems (IDPS).

TLS provides confidentiality and integrity. This prevents attackers from eavesdropping on traffic but attackers' payloads are sent to the server in the same way as normal traffic. Since TLS simply encrypts attackers' malicious data as it is sent to the server, it provides no protection from these attackers.

Network firewalls are a vital component for network defense, but they allow or deny traffic over a particular port. By necessity, web applications need to allow traffic through specified ports and attackers send their payloads using the same ports. Application gateways, even with careful configuration and self-learning, are also generic solutions to typical problems and IDPS has no real concept of good and bad application usage.

Additionally, some of these conventional defenses try to replicate part of the application logic but they can, at best, only do this partially. This external logic then needs to be verified, maintained, protected and managed, adding another often significant overhead to operational costs.

The generality of conventional defensive measures does mean they can be applied to numerous applications, often with little or no tuning, with obvious benefits such as cost and time to deploy. However, a significant challenge for this strategy is a lack of context. Detection of known attacks will almost certainly fail to detect a custom attack looking to exploit a weakness within an application. A different approach is required as conventional defenses are not working.

Application Defensive Measures

The right way to detect advanced attacks is within the applications themselves, by adding application-specific security controls that detect malicious activity.

Consider the analogy of control instrumentation in an industrial production process. Measurement points for temperature, mass, velocity, etc. are added to quantify key aspects in real-time. This information may be used in localized control, but is usually integrated into an overall process control system, which aggregates signals from many sensors, and uses various control mechanisms to react to changes and maintain a desired state. Unfortunately, most current software applications are like industrial processes without control instrumentation—there are no sensors and no intelligence about what is occurring—they are blind to what is happening internally.

But unlike conventional defensive measures, software applications have full knowledge about the business logic and the roles and permissions of users—each application can make informed decisions about misuse, and identify and stop attackers with a very low false-positive rate. An additional benefit is that this context-aware analysis occurs in real-time. This is an application-specific approach, not a generic one, because the controls are related and integrated within the application, and can be selected based on an assessment of risks specific to the process and data.

Similar ideas already exist in some software, but these are usually implemented as isolated processes and some may be undertaken reactively to events, or performed largely in a manual way. The concept described in this article focuses and formalizes this approach. It is about implementing proactive measures to add instrumentation and controls directly into an application in advance so that events are centrally analyzed and responded to.

Attack-Aware Detection

Once sensors have been deployed within the application code, they transmit detected event data to the analysis engine. This is then responsible for evaluation of individual events and determination of an attack. Once an attack has been recognized, the engine selects and executes an appropriate response.

Software applications then become attack-aware and it is possible to undertake proactive defense against currently unknown threats. Applications are the appropriate place to detect many types of attacks given the wealth of information available to them.

Normal and Malicious Behavior

The approach relies on being able to differentiate between normal and malicious behavior. That is one of the biggest hurdles when security controls (attack detection mechanisms) are added outside the application; it is difficult to distinguish user intent and this leads to a higher rate of false positives.

The ability to differentiate between normal behavior, suspicious behavior and attacks (and to do it with a high degree of granularity) can also be an improvement in the usability of human interfaces of the software applications. Application developers know which inputs allow keyboard entry, which have client-side validation, which should not be altered by a user and which are from trusted systems.

Evasion and Unknown Attacks

A common attack technique is using obfuscation to bypass black list inspection performed by security controls outside of the application. Integrating attack detection within the application solves this problem since all data is fully canonicalised. At this point it is simple for the application to inspect the user-submitted data and determine if it is malicious because the data is no longer obfuscated.

Detection within the application also allows insight into unknown attacks. Detection points can be designed to capture activity that could only occur outside the normal flow of the application. For example, a user would never accidentally trigger a detection point monitoring the use of the HTTP POST method where only the GET method is expected. In this example, the detection point can provide alerts that most likely represent attackers probing for weaknesses within the application.

OWASP AppSensor Project

OWASP AppSensor was developed under the OWASP Season of Code 2008 [2] by Michael Coates. The result is a ground-breaking work defining the AppSensor [3] concept, available as a printed book [4] and as a free PDF download [5]. Since its initial publication, a reference implementation in Java [6], a demonstration pilot [7], and several presentations and videos [8] have been developed.

Detection Point Type	Code	Name
Signature	RE	Request Exceptions
	AE	Authentication Exceptions
	SE	Session Exceptions
	ACE	Access Control Exceptions
	IE	Input Exceptions
	EE	Encoding Exceptions
	CIE	Command Injection Exceptions
	FIO	File IO Exceptions
	HT	Honey Trap
Behavioral	UTE	User Trend Exceptions
	STE	System Trend Exceptions
	RP	Reputation

Table 1: AppSensor guidance lists over 50 example detection point definitions grouped into these 12 categories; additional custom detection points are also often required.

Like an industrial process control system instrumentation engineer, developers have a whole range of sensors available. Most application sensors are located within the application code itself, but AppSensor also has the notion of inputs from other "external" devices. As an example, the software could alter its security posture based on external threat ratings such as the United States Armed Forces Defense Readiness Condition. AppSensor defines over 50 detection points [9] (Table 1) for monitoring. The list is not definitive, but is a starting point as most applications will need to consider adding custom detection points at various points deemed important.

AppSensor works best within the authenticated portion of an application where the user's identity is known and can be blocked if malicious, but it is also possible to apply the principles to other areas. Although the concepts can be retrofitted to existing software, they are most-easily built in during software development and the AppSensor document has guidance on detection point considerations, determining the malicious intent, monitoring system trend events and implementation guidance. However, the planning stages are probably the most time-consuming aspect of implementing AppSensor. If threat modeling is already being used in your software development lifecycle, this may be a natural location to determine many of your application specific detection points. Additional guidance [10] is available for assistance with this stage and includes a step-by-step description of how to plan implementation. Adoption can be encouraged, and the benefits realized by building AppSensor requirements into project specifications. Retrofitting existing software can also be undertaken, by building separate code libraries and calling these from within existing routines. The OWASP Enterprise Security API (ESAPI) [11] security control library can be used directly, or as a model for custom code library development.

Response Type	Examples
Allow	Log events only
Deny	Block IP address

Table 2: Conventional defensive measures may only offer a binary choice such as logging or blocking.

Rich Responses

Once the detection points have been determined, thresholds are set for each point, or group of points, which then act like tripwires. In AppSensor, an analysis engine monitors the detection events and determines when and what response is appropriate according to these specified event thresholds. Conventional defenses offer only binary response options such as allow/deny or log/block (Table 2). They are often limited to a single dimension—the particular network connection contravening a rule.

The AppSensor concept describes a wide spectrum of possible responses [12], and when combined with the very low false-positive detection rate, defenses do not need to be configured in detection-only mode for fear of preventing valid traffic. This provides a two-dimensional view, where responses may be directed not only against a single connection or a single user, but now also against a whole group (e.g., based on location, role, or behavior), or against all users. Table 3 lists some common types of response.

When behavioral aspects over time (a request/response, a session, a time period) are included, the application becomes capable of rich three-dimensional responses. The behavioral aspects can span beyond application restarts, upgrades and even the application lifecycle from deployment to disposal, since data from other systems that pre-date the application may be incorporated.

Cross Integration

As mentioned above, AppSensor can make use of information from other systems and devices to contribute to its pool of information for attack detection, to provide greater value. External systems may also be able to contribute to AppSensor's

Response Type	Examples
Logging Change	Full stack trace of error messages logged Record DNS data on user's IP address
Administrator Notification	Visual indicator displayed on an application monitoring dashboard Audible alarm in the control room
Other Notification	Signal sent to upstream network firewall, application firewall (e.g. XML, web) or load balancer Alert sent to fraud protection department
Proxy	Requests from the user invisibly (from the user's perspective) passed through to a hardened system Request are proxied to a special honeypot system which closely mimics or has identical user functionality
User Status Change	Change Increase data validation strictness for all form submissions by this citizen Reduce the number of failed authentication attempts allowed before the user's account is locked
User Notification	On-screen message about data validation issues Message sent by email to the registered email address to inform them their password has been changed
Timing Change	File upload process duration extended artificially Add fixed time delay into every response
Process Terminated	Discard data, display message and force user to begin business process from start Redirection of an unauthenticated user to the log-in page
Function Amended	Limit on feature usage rate imposed Additional validation requirements
Function Disabled	Web service inactivated Content syndication stopped
Account Logout	Session terminated and user redirected to logged-out message page Session terminated only (no redirect)
Account Lockout	User account locked permanently until an Administrator resets it One user's IP address range blocked
Application Disabled	Website shut down and replaced with temporary static page Application taken offline
Collect Data from User	Deploy a Java applet to collect remote IP address Deploy JavaScript to collect information about the user's network

Table 3: AppSensor provides the ability to have a much richer range of responses; the guidance lists 14 types of response. Specific examples for each type show how the types can be customized for each application's context.

decision-making processes. For example behavioral information can be fed into stochastic systems for statistical analysis, which may be the single most important pattern against unknown attacks to date. This defense against the unknown attacks is a giant benefit that can not be achieved using anything else, anywhere, at any price.

As well as consuming data, AppSensor can pass back intelligence to other systems. This might be used as part of the real-time response:

- Locking a user's single-sign-on account
- Removing some permissions for a user from a physical access control system
- Setting a warning flag about a customer
- Blocking a session ID or IP address at a network firewall.

Equally, data can be federated to other systems such as Security Information and Event Management (SIEM), Intrusion Detection Systems and fraud monitoring systems. One adopter in particular uses AppSensor heavily integrated with SIEM.

Information Insight and Additional Value

AppSensor provides insight into whether, and how, attacks are occurring. A pilot implementation has also demonstrated the value this approach has to containing the damage caused by application worms. In production applications, system trend detection points have been used to identify changes in a function's usage, alert administrators and ultimately disable the function. While this does not remove an infection, it stops a worm from spreading, limits the extent of data requiring clean-up, and may allow the remainder of the application to continue functioning, which could otherwise have had to be shut down. Another benefit is that AppSensor provides useful security metrics, currently in absentia.

Applicability

Software applications must be built securely in the first place. If issues like authentication, session management, authorization, etc. are broken already, it will not be possible to implement the approach. Self-defense goes beyond the implementation of security services, but there does appear to be growing interest in this area. For example, the DoD recently announced [13] that it is planning to spend $500 million to research new cyber security technologies including "active defenses"—technologies that detect attacks and probes as they occur.

AppSensor is a comprehensive proactive, rather than reactive, approach that can be applied to applications in many situations. It reduces the risk of unknown vulnerabilities being exploited by identifying and containing users conducting malicious activity that is often the precursor to an attack. It greatly increases the visibility of suspicious events and actual attacks. This can provide additional information assurance benefits:

- Reduced security risks to data and information systems
- Improved compliance
- Reduction in the consequences of data breaches.

In turn, these can provide improved service levels and resilience that have wide applicability. Some types of organizations may also achieve competitive advantage.

Although these concepts are already deployed in production environments, the AppSensor team is considering ways to improve the analysis engine including the use of stochastic methods, increasing coverage in ESAPI, integration with other systems and building the concepts into software frameworks and libraries so the implementation overhead is reduced. The team would also like to investigate how AppSensor could be applied in an agile/test-driven development environment.

Conclusions

All types of organizations have very powerful systems with access to mission-critical data, but generally have no idea if, when or how applications are under attack. This is the crux of the problem: critical data on critical systems and little to no visibility of the health of the overall environment. This must change. The concepts embodied by OWASP AppSensor present a solution to this issue. By creating applications that are attack-aware and able to respond appropriately, the assurance of information systems is increased.

Attack detection and response actions can be integrated into any application to greatly increase the overall security of the application. At a minimum, this technology provides greatly needed insight into the current attacks and malicious activities occurring within the application. When properly tuned, the attack-aware application will be able to identify and defend against malicious users in real-time. This technology must be implemented into mission-critical applications as we continue to see sensitive data and controls moving to systems that are subjected to a constant stream of malicious attacks.

Acknowledgements

The authors would like to acknowledge the support and collaboration of everyone who has contributed their ideas and time to the AppSensor project, and to the OWASP Foundation for providing the initial Summer of Code funding in 2008.

ABOUT THE AUTHORS

Colin Watson's work involves the management of application risk, building security and privacy into software development processes and keeping abreast of international legislation and standards. He holds a BSc in Chemical Engineering from Heriot-Watt University in Edinburgh, and an MSc in Computation from the University of Oxford. He contributes to a number of OWASP projects and is a member of the OWASP Global Industry Committee. Colin is a consultant and co-founder of Watson Hall Ltd.

E-mail: colin.watson@owasp.org

Continued on next page.

ABOUT THE AUTHORS

Michael Coates has extensive experience in application security, security code review and penetration assessments. He holds an MSc in Computer Security from DePaul University and a BSc in Computer Science from the University of Illinois. Michael is the creator and leader of the AppSensor project, a contributor to the OWASP Top 10 and a frequent speaker at security conferences. He is in charge of Mozilla's Infrastructure Security team responsible for the security lifecycle of web applications, and the protection of infrastructure and sensitive user data.

E-mail michael.coates@owasp.org

John Melton is a software architect and designer with particular knowledge of security for online systems. He is the AppSensor project's development lead, and has been responsible for most of the example code developed. He holds an MSc in Information Security and a BSc in Computer Science, both from the University of North Carolina at Charlotte. John is a Senior Information Security Engineer at Wells Fargo and lectures at UNC Charlotte.

E-mail john.melton@owasp.org

Dennis Groves is the co-founder of OWASP. He is a well-known thought leader in application security whose work focuses on multidisciplinary approaches to information security risk management. He holds an MSc in Information Security from the University of Royal Holloway, University of London. He is currently an expert for the UK mirror of ISO subcommittee 27, WG4.

E-mail dennis.groves@owasp.org

The OWASP Foundation
9175 Guilford Road Suite #300
Columbia, MD 21046
Telephone: 301-275-9403
Fax: 301-604-8033

REFERENCES

1. The Open Web Application Security Project (OWASP). About the Open Web Application Security Project. <http://www.owasp.org/index.php/About_OWASP>
2. Ibid. Summer of Code 2008. <http://www.owasp.org/index.php/OWASP_Summer_of_Code_2008>
3. Ibid. AppSensor Project, <http://www.owasp.org/index.php/Category:OWASP_AppSensor_Project>.
4. Coates, Michael. AppSensor - Detect and Respond to Attacks from Within the Application, v1.1, OWASP. Paperback, 48 pages, Lulu. <http://www.lulu.com/product/paperback/owasp-appsensor/4520003>
5. Ibid. AppSensor - Detect and Respond to Attacks from Within the Application, v1.1, OWASP. <https://www.owasp.org/images/2/2f/OWASP_AppSensor_Beta_1.1.pdf>
6. OWASP. AppSensor Developer Guide. <http://www.owasp.org/index.php/AppSensor_Developer_Guide>
7. Ibid. AppSensor Live Tutorial. <http://www.defendtheapp.com/>
8. Ibid. AppSensor Media. <http://www.owasp.org/index.php/OWASP_AppSensor_Project#tab=Media>
9. Ibid. AppSensor Detection Points. <http://www.owasp.org/index.php/AppSensor_DetectionPoints>
10. Watson, Colin. AppSensor Implementation Planning Workbook, OWASP, December 2010. <http://www.owasp.org/index.php/File:Appsensor-planning.zip>
11. OWASP. Enterprise Security API. <http://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API>
12. OWASP. AppSensor Response Actions. <http://www.owasp.org/index.php/AppSensor_ResponseActions>
13. Ratnam, Gopal and King, Rachael. "Pentagon Seeks $500 Million for Cyber Technologies", Bloomberg, 15 February 2011. <http://www.bloomberg.com/news/2011-02-15/pentagon-seeks-500-million-for-cyber-research-cloud-computing.html>

CROSSTALK
March/April 2011 The Journal of Defense Software Engineering Vol. 24 No. 2
THE RUG SOFT
CROSSTALK
May/June 2011 The Journal of Defense Software Engineering Vol. 24 No. 3
PEOP SOLUTIO SOFTW PROBL
CROSSTALK
July/August 2011 The Journal of Defense Software Engineering Vol. 24 No. 4
MANAGEMENT PRACTICES for QUALITY PRODUCTS