OWASP AppSensor - CrossTalk

Article submission for the "Protecting Against Predatory Practices" themed September/October 2011 edition of CrossTalk, The Journal of Defense Software Engineering.

Draft v0.8 (24th March 2011)

Title

Building Real-Time Defenses into Software Applications

Abstract

Attack-aware software applications respond in real time to suspicious and malicious events with a very low false positive detection rate. 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, tampering and theft. The OWASP AppSensor Project has developed a methodology, documentation, example code and pilot demonstration for this application-specific attack detection and response.

Copyright

Building Real-Time Defenses into Software Applications

Introduction

Information systems and data are being **targetted** by advanced attackers who are skilled, motivated and have excellent tools. They may be backed by organized crime, governments or commercial enterprises. Attackers will relentlessly attempt to access sensitive and secret data available to software applications by finding and exploiting vulnerabilities in the applications themselves.

In this article we discuss why traditional defenses are not a solution to these types of advanced attack, and explain an initiative from the Open Web Application Security Project (OWASP)[1] which addresses the problem.

Traditional Defensive Measures

A fundamental starting point for secure software applications is to build security in at all stages of the development life-cycle. We need robust code designed, developed, configured and operated on hardened operating systems. This does not mean the software is impregnable; unknown vulnerabilities almost certainly exist, and attackers examine and probe applications to find them. Once they have identified one or more vulnerabilities, they need to discover ways to exploit them individually, or in combination. It is very hard to detect these custom attacks using traditional defensive measures. One reason is that traditional defenses either do not work at the application layer or they are designed with standard signatures to look for known common bad patterns. This may detect generic attacks, but will almost certainly fail to detect a custom attack looking to exploit a weakness within the application.

Some techniques often thought to defend applications include using transport layer security (TLS/SSL), network firewalls, application gateways (e.g. web application firewalls, XML appliances, cross domain guards) and network/host intrusion detection and prevention systems (IDPS).

TLS is not a complete solution to the services required for a secure application. In addition to confidentiality and availability, minimum security standards dictate some kind of data integrity, along with authentication and authorization. TLS provides some degree of data confidentiality and integrity in transit and limited assurance in the identity of one or more of the parties depending upon how it has been configured. Most deployments use a single certificate model as e-commerce applications often perform their identity checks at payment; this however subverts a very important design criteria of the TLS standard which derives it security strength from a dual certificate exchange. However there is no inspection of the data itself — it simply passes on what was sent. Attackers' payloads are sent to the server in the same way as normal traffic. In the end, since TLS simply encrypts attackers' malicious data as it is sent to the server, it provides no protection to prevent an attacker from attacking the website.

Network firewalls are a vital component for network defense, but they allow or deny traffic using a particular port. By necessity web applications need to allow traffic through specified ports (often TCP ports 80 or 443). Attackers of course send their payloads using the same ports. And it is for this very reason, that it is commonly believed that the perimeter defence model is dead. Application gateways are generic solutions to typical problems. For example, even with careful configuration and self-learning, most web application firewalls are operated in detection-only mode, and they have no insight into business logic type of attacks. Like network firewalls, IDPS has no real concept of good and bad application usage.

Additionally, some of these traditional defenses try to replicate some 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.

While many traditional ~~techniques~~defensive measures are generic, this means that they can be applied to ~~many~~numerous applications, often with little to no tuning~~. This~~ which has obvious benefits (cost, time to deploy, etc.). However, a significant challenge ~~for many of these techniques~~ is a lack of context. Context is what allows applications to have intimate knowledge of both data and the users

accessing that data. In today's world, our applications are the gateways to our data and attackers are constantly probing to determine weaknesses in our systems. Proactively detecting attackers via attack-aware applications can significantly increase the assurance of our systems.

Real-Time and Proactive Defensive Measures

Consider the analogy of control instrumentation in an industrial production process. Measurement points for temperature, pressure, mass, velocity and volumetric, etc are added to quantify the state of key aspects over time. This information may be used in localised control (e.g. a float switch in a liquid storage vessel), but more often is integrated into an overall process control system, which aggregates signals from many sensors, and uses feed-back and feed-forward control mechanisms to react to changes and maintain a desired state.

Unfortunately, most current software applications are like industrial processes without this control instrumentation — there are no sensors and no intelligence about what is going on — they are blind to what is happening internally. Yet applications have access to powerful processing and storage capabilities in their supporting hardware and software systems.

The right way to detect advanced attacks is within applications themselves, by adding application-specific security controls which detect malicious activity. The application has full knowledge about the business logic and the roles & permissions of users — it can make informed decisions about mis-use, and to 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 a 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.

Once sensors have been deployed within the application code (and possibly external sensors), detection and response are the next steps. The sensors themselves send any events they detect to the analysis engine. The engine is then responsible for evaluation of individual events and determination of an attack. Once an attack has been recognized, the analysis engine then must determine an appropriate response and execute that response.

Within the Application is Best

By building attack detection measures within software applications, they become attack-aware. By being attack-aware 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 the application. Within the application:

- data is fully available (e.g. decrypted)
- the context is known
- the business logic is known (e.g. purposes, paths and sequencing)
- information about the user is fully accessible (e.g. user identity, role, status, previous actions)
- program flow can be altered in real time

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. Some examples of these include:

- counting multiple failed authentication attempts to lock a user account
- detecting use of the TRACE HTTP method to block requests
- checking the IP address during a session and terminating the session if the IP address changes
- logging unexpected requests
- investigating suspicious events at a later date.

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 all these events (and more) are centrally analysed and responded to.

Normal and Malicious Behavior

The attack-aware concept 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. When detection is undertaken in the application itself, the application has access to the complete context of a request as well as information about the user. This means the application can insert precise detection points that identify attempts to bypass access control, bypass client-side input validation, call processes in the wrong order, etc. These detection points capture activity that is application specific and clearly malicious — none of these events are possible by accident. This knowledge is not available to generic security defenses outside of the application or on the network.

This fine-grained ability to differentiate between normal behavior, suspicious behavior and attacks can also be an improvement in the usability of human interfaces of the software applications. The application knows which inputs allow keyboard input, which have client-side validation and which should not be altered by a user. A traditional defense such as a gateway device applying a whitelist derived by auto-learning might block a password which has a trailing line feed character appended. The application might consider this invalid, but not consider it suspicious since the extra character could be the result of a straightforward copy & paste action. Thus the application does not block normal usage unnecessarily.

Evasion and Unknown Attacks

A common attack technique is to use obfuscation to bypass black list inspection performed generic application defenses outside of the application. Another benefit of integrating attack detection within the application is that all data has been fully canonicalised. At this point it is much easier for the application to inspect the user submitted data and determine if the data is malicious.

In addition, detection within the application allows insight into unknown attacks. Many detection points are designed to capture activity that can only occur outside the normal flow of the application. In other words, a user could never accidentally trigger the detection point (e.g. a web page receives a POST message but is only designed for GET). In these scenarios, the detection points can provide alerts to odd behavior that could represent attackers probing for a new type of weakness within the application.

AppSensor Project

The concept described above was initially developed under the OWASP Season of Code 2008[2] by Michael Coates. The output was a document defining the AppSensor[3] conceptual framework, available as a printed book[4] and free PDF download[5]. Over the last two years a reference implementation in Java[6], further guidance[7], demonstration pilot[8], presentations and videos[9] have been created to support the initiative.

Like in an industrial process control systems, a whole range of sensors can be considered for incorporation by the instrumentation engineer. Most of the equivalent application sensors are located within the application code itself, but AppSensor also has the notion of inputs from other "external" devices. In application software, these external inputs could come from diverse sources such as:

- user profiling
- reputational services (e.g. credit checking, risk tolerance, threat level)
- fraud monitoring system
- CRM system
- file system integrity monitoring
- network security devices (e.g. network and application firewalls).

As an example, the software could alter its security posture based on an external threat level rating such as the United States Armed Forces Defense Readiness Condition (DEFCON).

Some detection points identify specific events defined as a generic signature; others consider behavior over time. Furthermore, it is possible to detect and differentiate between both suspicious and attack events, providing additional insight into how an application is being used, and abused. Some detection points also look at the results, or output, rather than user inputs. For example the number of records returned by a database or XPath query or the integrity of relational database tables or log files may be evaluated to determine whether an attack has occurred.

AppSensor has defined over 50 detection points[10] grouped into twelve categories (Table 1) reflecting the aspect being monitored. The detection points range from signature type detection points which typically look for particular events within a user's single request/response cycle, to behavioral type detection points which assess events over longer time periods (e.g. a single user session, a finite time period, the life of the application). The list is not definitive; but is a suggested starting point since custom applications will need to consider carefully adding detection points specific to the application at various points deemed important to the custom business logic within the application.

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

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

Full descriptions, examples and considerations to maintain a low false positive detection rate of each detection point have been documented. Implementers are encouraged to define additional custom detection points where the suggested ones are insufficient for the logic involved.

Thresholds are set for each detection point, or group of detection points which then act like tripwires. There is an analysis engine which monitors the detection events and determines when, and what, response is appropriate according to the user-specified event thresholds.

Rich Responses

Traditional defenses offer 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.

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

Response Type	Examples
Allow	Log events only
Deny	Block IP address

The AppSensor concept describes a wide spectrum of possible responses[11], 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. Inside the application 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 the most common responses, from those hidden from the user, to those which the user may be aware of and those which actually interfere with an action.

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

Response Type	Examples
Logging Change	Capture sanitised request headers and response bodies
	Full stack trace of error messages logged
	Record DNS data on user's IP address
	Security logging level changed to include 'informational' messages
Administrator Notification	Email alert sent to everyone in the administration team
	SMS alert sent to the on-call administrator
	Visual indicator displayed on an application monitoring dashboard
	Audible alarm in the control room
Other Notification	Broadcast event to SIEM
	Signal sent to upstream network firewall, application firewall (e.g. XML, web) or load balancer
	Alert sent to fraud protection department
	Record added to server event log
	Event highlighted in a daily management report
	Email alert to staff member's manager
	Proactive entry added to customer support system
User Status Change	Internal trustworthiness scoring about the user changed
	Reduce payment transfer limit for the customer before additional out-of-band verification is required
	Reduce maximum file size limit for each file upload by the forum user
	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 (ASR-K below)
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 Notification	On-screen message about mandatory form fields
	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	Extend response time for each failed authentication attempt
	File upload process duration extended artificially
	Add fixed time delay into every response
	Order flagged for manual checking
	Goods despatch put on hold (e.g. despatch status changed)
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
	Redirection to home page
	Display other content (i.e. terminate process but display the output of some other page without redirect)
	Redirection to a page on another website
Function Amended	Limit on feature usage rate imposed
	Reduce number of times/day the user can submit a review
	Additional registration identity validation steps
	Additional anti-automation measures (e.g. out-of-band verification activated, CAPTCHA introduced)
	Static rather than dynamic content returned
	Additional validation requirements for delivery address
	Watermarks added to pages, images and other content
	Additional human interactive proof challenges added due to the number of incorrect guesses of CAPTCHAs outnumbering the correct guesses by more than a certain factor (e.g. Token bucket idea)
Function Disabled	Add friend' feature inactivated
	'Recommend to a colleague' feature links removed and disabled
	Document library search disabled
	Prevent new site registrations
	Web service inactivated
	Content syndication stopped
	Automated Direct Debit system turned off and manual form offered instead
Account Logout	Session terminated and user redirected to logged-out message page
	Session terminated only (no redirect)

Account Lockout	User account locked for 10 minutes
	User account locked permanently until an Administrator resets it
	One user's IP address range blocked
	Unauthenticated user's session terminated
Application Disabled	Website shut down and replaced with temporary static page
	Application taken offline
Collect Data from User	Deploy additional browser fingerprinting using JavaScript in responses
	Deploy a Java applet to collect remote IP address
	Deploy JavaScript to collect information about the user's network

Thus for example, when an employee activates a small number of detection points in an application providing access to secret information, a response could be to alert supervisors and either decrease response times or proxy requests to a less sensitive system. A smart meter might disable a diagnostic function which appears to have been compromised, whilst continuing to allow electricity supply and billing to the home owner. When suspicious behavior is detected , a banking application might reduce a customer's payment transfer limits or hold the transaction temporarily until further verification of authenticity can be made.

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 life cycles from deployment to disposal, since data from other systems which pre-date the application may be incorporated.

Cross Integration

As mentioned above, AppSensor can potentially make better use of information from other systems and devices to contribute to its pool of information for attack detection, increasing the value of those other systems. Other systems may be able to contribute to AppSensor's decision making processes. For example behavioral information can be fed into stochastic systems for statistical analysis, which may be useful in identifying user error from malicious behavior. Something that occurs very infrequently but causes an exception to be thrown may actually be down to human error, such as a typo, or other anomaly, whereas hundreds of the same event happening at once may indicate a malicious behavior. Hundreds of people are very unlikely to be making the very same error at the very same time. Additionally, this further reduces false positives since stochastic methods 'tune' the system for outliers.

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 in a CRM
- blocking a session ID or IP address at a network firewall.

Equally, data can be federated to other systems such as SIEM, IDS and fraud monitoring systems.

Information Insight and Value

Applications contain considerable knowledge about users and their activities. Without AppSensor so much intelligence is being wasted. System owners are left with tools and devices which have to guess about what is going on. 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. System trend detection points can be used to detect large changes in a function's usage, alert administrators and ultimately disable the function. While this does not remove the infection, it stops the worm from spreading, limits the extent of data requiring clean-up, and may allow the remainder of the application to continue functioning, which may otherwise have had to be stoped.

And of course if this is a public-facing commercial application, some of these data would be valuable to those interested in understanding and improving operational objectives relating to metrics such as those for availability, usability, reducing drop-out rates, conversion ratios, etc. Another key benefit, is that this give security useful metrics, currently in absentia.

Applicability

The recent announcement[12] from the U.S. Defense Department 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 - suggests there is growing interest in this area.

OWASP AppSensor is a comprehensive proactive approach, rather than reactive, which can be applied to applications ~~throughout the enterpris~~in many situations~~e~~. 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 and, for ~~commercial~~some organizations, competitive advantage.

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 detection points correctly.

Although AppSensor works best within the authenticated portion of an application, since a unique user identity is known and can be blocked if malicious, it is also possible to apply the principles to other areas. In the latter, the range of "normal behavior" may be wider, the identity and location of users may be harder to pinpoint and some detection points may no longer be necessary, but the same benefits are possible.

Existing AppSensor documentation already includes 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 modelling is already being used in your software development life-cycle, this may be a natural location to determine many of your application specific detection points but recently additional guidance[13] has been created to assist with this stage.

AppSensor has a very low false positive detection rate. The implementation must ensure that this is not compromised by adding inappropriate detection points, or implementing them in a way which leads to a greater number of false positives. The method presented also tries to build in consideration of business operations and usability, so that not only is a low false positive rate maintained, but processes are not unduly disrupted and the users are not subjected to difficulties through simple human error. In other words, building in a degree of human fault tolerance. The guidance also addresses the risks of denial of service, by working through how the application might respond to a range of attack scenarios and the effects on users. The range of responses available and ability to segment responses by user role mitigates this risk.

Conclusions

Right now, we have very powerful systems with access to mission critical data, but we generally have no idea if, when or how our applications are under attack. This is the crux of the problem we face; critical data on critical systems and little to no visibility of the health of the overall environment. This must change. The concepts embodied by AppSensor present a solution to this issue. The AppSensor

concept says that the application must also defend itself, as well as provide all of its own security services.

By creating applications that are attack-aware and able to respond appropriately, the assurance of our systems is increased. In addition, we regain operational visibility of the applications that hold our most critical data.

...When combined with stochastic methods, the sensitivity to unknown attacks can be increased further making OWASP AppSensor the single most important pattern against unknown attacks to date.

Restate the key benefits of OWASP AppSensor, ...

> Defense against the unknown attacks is a giant benefit that can not be had anywhere else at any price.
>
> ???

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

Colin Watson's work involves the management of application risk, building security and privacy into systems development and keeping abreast of relevant 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.

Email colin.watson@owasp.org

Michael Coates

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. As web security lead at Mozilla, Michael protects web applications used by millions of users each day.

Email michael.coates@owasp.org

John Melton

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.

Email ~~jtmelton@gmail.com~~john.melton@owasp.org

Dennis Groves

Dennis takes a multidisciplinary approach to risk management and is a thought leader in the web application security space. He is particularly interested in risk, randomness, and uncertainty. He holds an MSc in Information Security from Royal Holloway, University of London. As a co-founder of OWASP, his contributions include writing the first version of the OWASP Guide which is the the de-facto standard for securing web applications, and is now a reference document in PCI DSS.

Email dennis.groves@owasp.org

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 Implementation Planning Workbook. <http://www.owasp.org/index.php/File:Appsensor-planning.zip>

8 Ibid. AppSensor Live Tutorial. <http://www.defendtheapp.com/>

9 Ibid. AppSensor Media. <http://www.owasp.org/index.php/OWASP_AppSensor_Project#tab=Media>

10 Ibid. AppSensor Detection Points. <http://www.owasp.org/index.php/AppSensor_DetectionPoints>

11 Ibid. AppSensor Response Actions. <http://www.owasp.org/index.php/AppSensor_ResponseActions>

12 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>

13 Ibid. AppSensor Implementation Planning Workbook. <http://www.owasp.org/index.php/File:Appsensor-planning.zip>