

Attack Detection & Response with OWASP AppSensor

An Implementation Planning Workbook

V0.3 - 3rd August 2011

Colin Watson, [Watson Hall Ltd](#)

Abstract

This workbook provides a methodology to plan AppSensor implementations. The planning stage includes sensor selection and positioning, and determination of the appropriate type of response to block or mitigate attacks based on an analysis of business risk, process criticality and user experience requirements. New practical templates and charts are provided to help embed application intrusion detection into development and procurement processes. A lightweight implementation is also described for organizations wishing to pilot AppSensor in their applications.

© 2010-2011 OWASP Foundation

This document is licensed under the [Creative Commons 3.0 Attribution-ShareAlike](#) License



Credits

The AppSensor concept was created by Michael Coates. Through his management and encouragement the AppSensor project has grown into a much larger project supported by a broad group of people. Discussions on the AppSensor mailing list and development of real Java code have contributed greatly to the development of the ideas in this document. Insights from Michael Coates, John Melton and Ryan Barnett have been particularly thought-provoking.

Feedback

Your feedback on the document and experience from implementing AppSensor would be greatly welcome.

Disclaimer

This document is in a state of flux while it is being applied in practical engagements. Other contributors to the AppSensor project have not been involved and therefore all mistakes and errors are the author's own.

Organizations should use the methods in this document in the context of their own business processes and risk assessments. The success of implementation comes down to many details.

Changes in v0.3

Detection points RE7 and RE8 (suggested by Ryan Barnett in November 2010) added to tables and charts.

Response action ASR-P for "no response" added to text.

Minor formatting and grammar updates.

Contents

Introduction	4	Conclusion	50
Methodology steps:		Quickstart implementation plan	51
1. Preliminary requirements	8	List of Tables	53
a) Application risk assessment	9	List of Figures	56
b) Secure coding	13		
c) Application logging	14	Appendices	58
2. Detection point selection	15	A: Detection point categorization	59
a) Categorization	16	B: Detection point cross reference with attacks, weaknesses and risks	63
b) Requirements	20	C: Functional grouping of detection points	68
<i>Risk classification</i>	21	D: Example high-level data flow diagrams annotated with detection points	76
<i>Threat assessment</i>	22	E: Detection point tuning analysis considerations	80
<i>Class categorization</i>	22		
<i>Custom types</i>	26		
c) Model development	27		
d) Optimization	32		
<i>Sensitivity tuning</i>	32		
<i>Effects of related systems</i>	34		
<i>Model reduction</i>	35		
e) Code location	37		
f) Attack analysis	39		
3. Response action selection	40		
a) Strategic requirements	42		
b) Thresholds	43		
c) Model tuning	48		

Introduction

This document is a description of an AppSensor planning methodology to assess an application, and select application layer intrusion detection and response. It is language/framework independent.

Attack detection and prevention with OWASP AppSensor

OWASP AppSensor defines a conceptual framework, methodology, guidance and example code to implement intrusion detection and automated response. AppSensor builds detection into the application to look for unacceptable malicious attacks. The idea is similar to the approach taken by banks to physical security. They don't just rely on strong walls and doors, but also have camera surveillance and alarm systems inside. In the same way, applications should have self-protection built in.

Over 50 detection points (like mini alarms or trip wires) are defined together with a number of response actions. Many malicious attacks are obvious and not "user error" such as receiving an HTTP POST request when expecting GET, or when headers are tampered with, or when a cross-site scripting (XSS) attack is submitted. Application usage behavior can be thought of as a continuum of unacceptable to acceptable behavior—AppSensor is concerned with identifying and responding to malicious events, beyond the range of normal user behavior:

Figure 0-1: The range of user behavior illustrating that malicious attacks are separate to normal application use



The following two video presentations, by the project leader Michael Coates, provide a greater insight into the concepts and benefits of using AppSensor:

- Automated Application Defenses to Thwart Advanced Attackers
<http://michael-coates.blogspot.com/2010/06/online-presentation-thursday-automated.html>
- Real Time Application Defenses - The Reality of AppSensor & ESAPI
<http://vimeo.com/15726323>

AppSensor's objectives

Organizations are concerned about protection of the application, the application's users and business & user data. AppSensor aims to reduce risks to these by detecting attackers probing or attacking the application, and stopping them before they can exploit a vulnerability. It has also been shown previously (see Automated Application Defenses to Thwart Advanced Attackers, page 2) how AppSensor can be used to contain the effects of an application worm. By intervening and blocking it becomes less economically feasible to attack the application. AppSensor does not care about the identity of who is attacking, when they are going to attack or where they are coming from.

Detection is undertaken at the application layer and, unlike other protection devices on the network, the application itself has access to the complete context of a request and information about the user. Input data are decrypted and canonicalised within the application and therefore application intrusion detection is less susceptible to advanced evasion techniques.

Embedding protection into applications

AppSensor is a comprehensive proactive, rather than reactive, approach which can be applied to applications throughout the enterprise. It reduces the risk of unknown vulnerabilities being exploited. 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 organizations, competitive advantage.

Implementation

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 and with the recent addition of further detection points and response actions, this document provides more information on the process and considerations to take.

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.

Although AppSensor works best within the authenticated portion of an application, 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.

Build on what you have already

AppSensor's detection point ideas are not necessarily novel, but build on common security principles. Some similar ideas may already exist in an application, but these will typically be 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.

AppSensor focuses and formalizes this approach. AppSensor 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. The application has full knowledge about the business logic and the roles & permissions of users—it can make informed decisions about mis-use, and identify and stop attackers with an extremely low false positive rate. It also does this in real time.

Additionally, AppSensor can potentially make better use information from other security devices to contribute to its pool of information for attack detection, increasing the value of those other systems.

Appsensor instrumentation analogy

Like the controls in a car's engine management system, or in heating boiler or furnace, or alarms inside a bank, we need to select and locate the sensors and control mechanisms appropriately.

A car engine management system will have many sensors... detectors for fluid levels, battery charge, engine speed, vehicle speed and acceleration, ignition timing, combustion, exhaust and tyre pressures, ambient, engine and exhaust gas temperatures, integrity of safety mechanisms, door lock status, entry alarm and isolator status, service history, performance data..., and so on. Instrumenting an application needs careful planning too.

Implementation methodology overview

It is necessary to build applications securely in the first place, and understand the risks the application faces. Once these are known, the appropriate AppSensor functions (detection points and response actions) can be selected and added to the application.

The proposed implementation methodology follows these *steps*:

1. Preliminary requirements
2. Detection point selection
3. Response action selection

The methodology does not identify which technologies should be used. It should be modified to suit each organization's business and development processes.

The *steps* are described in the following pages.

1. Preliminary requirements

- a) Application risk assessment
- b) Secure coding
- c) Application logging



1a) Application risk assessment

Before embarking on the implementation of AppSensor, decide what needs to be protected and with how much effort. This can normally be linked with the outputs from an existing risk assessment processes. The process is defined in SAMM:

Strategy & Metrics Level 2, Software Assurance Maturity Model (SAMM), v1.0

<http://www.opensamm.org/downloads/SAMM-1.0.pdf>

- Objective "Measure relative value of data and software assets and choose risk tolerance"
- Activity "Classify data and applications based on business risk"

For further information on how to undertake risk assessments, see these resources elsewhere:

- Memorandum M-04-04, Office of Management and Budget, US Government, 16 December 2003
<http://www.whitehouse.gov/sites/default/files/omb/memoranda/fy04/m04-04.pdf>
- Special Publication 800-30 Risk Management Guide for Information Technology Systems, NIST, July 2002
<http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf>
- FIPS PUB 199 Standards for Security Categorization of Federal Information and Information Systems, NIST, February 2004
<http://csrc.nist.gov/publications/fips/fips199/FIPS-PUB-199-final.pdf>
- An Introduction to Information System Risk Management, Steve Elky, SANS Institute, 2007
http://www.sans.org/reading_room/whitepapers/auditing/introduction-information-system-risk-management_1204
- Risk Rating Methodology, OWASP
http://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology
- Threat Risk Modelling, OWASP
http://www.owasp.org/index.php/Threat_Risk_Modeling
- Links to resources about Risk Analysis, Risk Assessment, Risk Management
<http://www.nr.no/~abie/RiskAnalysis.htm>

This process should provide much insight into the applications, but most importantly allows you to rank them based on your own criteria. The criteria may be from the organization's viewpoint, but it is worth also taking into account the value of the data and system from other perspectives such as its users, other parties and society.

Thus applications may be categorized 1-10, or AL1-5 or low, medium, high etc.

Figure 1-1: Two different organization's application portfolio risk classifications showing the number of applications in each
Risk classifications are organization-specific

Risk Classification		Risk Classification	
Critical	25	AL5	2
High	18	AL4	5
Medium	8	AL3	4
Low	10	AL2	8
		AL1	3

The following resources discuss portfolio risk ranking:

- Web Application Security Portfolios, Nick Coblentz, ISSA Journal, May 2009 and <http://nickcoblentz.blogspot.com/2009/06/repost-web-application-security.html>
- Application Portfolio Risk Ranking: Banishing FUD With Structure and Numbers, Dan Cornell, Denim Group http://www.owasp.org/index.php/Application_Portfolio_Risk_Ranking:_Banishing_FUD_With_Structure_and_Numbers

Nick Coblentz highlights the issue of common dependences. The applications may be isolated as shown in the figure below.

Figure 1-2: Example risk classification for five independent applications

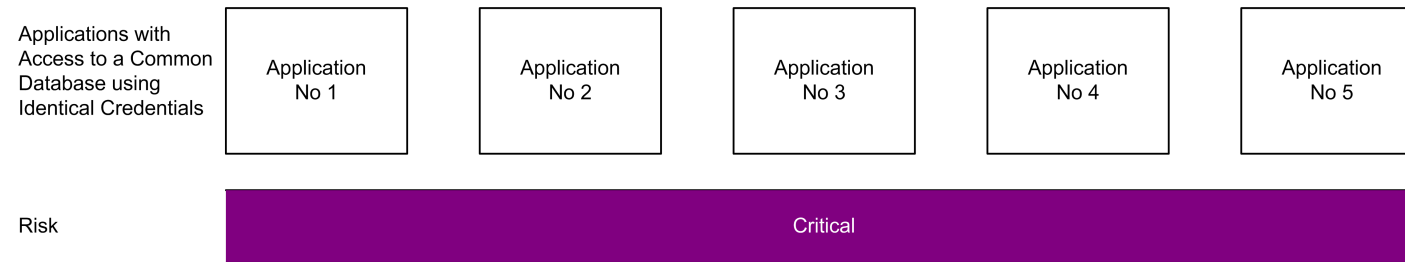
The applications are completely isolated from each other

Completely Isolated Applications	Application No 1	Application No 2	Application No 3	Application No 4	Application No 5
Risk	Low	Medium	Critical	Low	High

However, common dependencies such as shared components, identical data access, common hosting or inter-related back-end systems may mean all applications needs to be considered at the greatest risk classification.

Figure 1-3: Example risk classification for five related applications

The applications use a common database and share login credentials (poor practice)



An understanding of the dependencies and inter-relationships is necessary to ensure AppSensor detection points are selected and applied appropriately, and in the most efficient manner. In some cases, it may be possible to partition an application into sections, with different risk ratings, and this could be used to allocate AppSensor detection points in a more targetted manner.

One possibility to consider is whether the application can be partitioned into public areas, authentication, private areas for authenticated users and perhaps back-office functionality such as a web-based content management system or other website administration functionality. AppSensor defends against an attacker who might be able to find a vulnerability; for an unknown vulnerability, you do not know the likelihood nor impact, but you should know the exposure. Derive the impact from the risk assessment for the whole application.

In Figure 1-4, we have indicated the case where ease of exploitation is simple and the impact is severe. For an unknown vulnerability you may have to assume this worst-case scenario if some form of application manipulation or data disclosure might lead to a severe impact.

Figure 1-4: Example risk partitioning of an application

If each section is truly isolated from the others, the exposure will affect the resultant risk classification

Possible Application Partitions	Public (internet)	Authentication (internet)	Authenticated Users Only (internet)	Web Service (partner network)	Back-Office (internal network)
Exposure	Very High		High	Medium	Low
Ease of Exploitation	Simple				
Impact	Severe				
Risk	Critical		High	Medium	Low

However in most situations, it is more usual to treat the application as a single unit.

1b) Secure coding

AppSensor reduces the risk of an attacker finding an unknown vulnerability. AppSensor is not a defensive strategy for insecure applications. In fact using detection points assumes you already have in place secure coding practices such as input validation.

However an application is developed or acquired, the practices under the business functions defined in the Software Assurance Maturity Model are recommended:

- Governance
- Construction
- Verification
- Deployment

In particular, the host environment should be hardened and procedures in place for vulnerability management and incident response.

Additionally the information available in the OWASP Development, Code Review and Testing Guides and security verification requirements defined in the Application Security Verification Standard should be considered.

- Software Assurance Maturity Model
http://www.owasp.org/index.php/Category:Software_Assurance_Maturity_Model
- Application Security Verification Standard
<http://www.owasp.org/index.php/ASVS>

1c) Application logging

Web server logs are not application logs or firewall logs.

All types of security related events should be being recorded by an application. AppSensor will rely on some type of historical knowledge of previous events (detection point being activated), and this can be an extension of well-designed application logging facilities or a custom intrusion store.

Guidance on application logging is beyond the scope of this document, but the following resources are recommended:

- How to Do Application Logging Right
IEEE Security & Privacy Journal, Anton Chuvakin and Gunnar Peterson
<http://arctecgroup.net/pdf/howtoapplogging.pdf>
- Securosis Blog - Monitoring Up the Stack series
<http://securosis.com/blog/monitoring-up-the-stack-app-monitoring-part-1>
<http://securosis.com/blog/monitoring-up-the-stack-app-monitoring-part-2>
- OWASP AppSensorDemo2 Intrusion Store (Hyper Structured Query Language Database)
<http://code.google.com/p/appsensor/source/browse/#svn/trunk/AppSensorDemo2>
- OWASP ESAPI Java Edition documentation
<http://code.google.com/p/owasp-esapi-java/>
- Preventing Log Forging in Java
<http://www.jtmelton.com/2010/09/21/preventing-log-forging-in-java/>
- SP 800-92 Guide to Computer Security Log Management
NIST <http://csrc.nist.gov/publications/nistpubs/800-92/SP800-92.pdf>

Some further notes on the above are provided at <http://www.clerkendweller.com/2010/8/17/Application-Security-Logging>

For AppSensor, it is useful to be able to:

- ensure events can be grouped by request – multiple events may occur for a single request/response – e.g. log a unique request ID
- include details of which AppSensor detection points were activated if applicable (code and instance e.g. IE5-4054)
- include any AppSensor response actions taken and the final HTTP status code.

These might be added to the normal application security event logging, or be recorded in supplemental files/data stores.

2. Detection point selection

- a) Categorization
- b) Requirements
- c) Model development
- d) Optimization
- e) Code location
- f) Attack analysis



In a car engine, engineers have to decide which sensor instrumentation is installed in the engine and where to locate it. Applications need to be treated in a similar way. We need to select the detection points needed to achieve the desired protection requirements in the most cost-efficient manner.

We will consider the detection requirements to create a model, look at how to optimize this model and check it using attack analysis before considering the response actions in *step 3*.

The analysis is suitable both for consideration during procurement, as well as development processes. Outsourced development and services could be asked to implement AppSensor and provide access to the event data.

2a) Categorization

Firstly we will summarize the detection points under various categories. This will be an aid to selecting appropriate detection points in *step 2b*. The detection points specified in AppSensor documentation are summarised by exception type in Table 2-1 on the following page (source: http://www.owasp.org/index.php/AppSensor_DetectionPoints). We will use the identity codes forthwith, so this table will be our main reference.

It should also be noted that a few detection points detect an outcome/result of the request, rather than the request's content:

- IE5 Violation of Stored Business Data Integrity
- IE6 Violation of Security Log Integrity
- CIE2 Detect Abnormal Quantity of Returned Records

In some circumstances, RP3 Suspicious Client Side behavior might also be considered an outcome/result—perhaps some XSS occurs on the response page once rendered by the user's web browser.

The AppSensor documentation also categorizes the detection points as to whether they are signature-based or behavior based and this is summarised in Appendix A, Table A-1. This table also differentiates between detection points that monitor single users versus those that are concerned with all users of the application. We can see that STE1, STE2, STE3 and RP4 derive their information from the activities of multiple users.

The documentation also categorises detection points based on malicious intent:

- Suspicious events which could occur during normal user experience with site or browser or as the result of a non-malicious user error
- Attack event which are outside of the normal application flow, or requires special tools or requires special knowledge

The detection points are summarized in this manner in Appendix A, Table A-2.

Table 2-1: AppSensor Detection PointsListed by exception type, then alphabetically by ID from http://www.owasp.org/index.php/AppSensor_DetectionPoints

CATEGORY	ID	TITLE	CATEGORY	ID	TITLE
Request Exception	RE1	Unexpected HTTP Command	Input Exception	IE1	Cross Site Scripting Attempt
	RE2	Attempt to Invoke Unsupported HTTP Method		IE2	Violation Of Implemented White Lists
	RE3	GET When Expecting POST		IE3	Violation Of Implemented Black Lists
	RE4	POST When Expecting GET		IE4	Violation of Input Data Integrity
	RE5	Additional/Duplicated Data in Request		IE5	Violation of Stored Business Data Integrity
	RE6	Data Missing from Request		IE6	Violation of Security Log Integrity
	RE7	Unexpected Quantity of Characters in Parameter	Encoding Exception	EE1	Double Encoded Character
	RE8	Unexpected Type of Characters in Parameter		EE2	Unexpected Encoding Used
Authentication Exception	AE1	Use of Multiple Usernames	Command Injection Exception	CIE1	Blacklist Inspection for Common SQL Injection Values
	AE2	Multiple Failed Passwords		CIE2	Detect Abnormal Quantity of Returned Records
	AE3	High Rate of Login Attempts		CIE3	Null Byte Character in File Request
	AE4	Unexpected Quantity of Characters in Username		CIE4	Carriage Return or Line Feed Character in File Request
	AE5	Unexpected Quantity of Characters in Password	File IO Exception	FIO1	Detect Large Individual File
	AE6	Unexpected Type of Character in Username		FIO2	Detect Large Number of File Uploads
	AE7	Unexpected Type of Character in Password	Honey Trap	HT1	Alteration to Honey Trap Data
	AE8	Providing Only the Username		HT2	Honey Trap Resource Requested
	AE9	Providing Only the Password		HT3	Honey Trap Data Used
	AE10	Additional POST Variable	User Trend Exception	UT1	Irregular Use of Application
	AE11	Missing POST Variable		UT2	Speed of Application Use
	AE12	Utilization of Common Usernames		UT3	Frequency of Site Use
Session Exception	SE1	Modifying Existing Cookie		UT4	Frequency of Feature Use
	SE2	Adding New Cookie	System Trend Exception	STE1	High Number of Logouts Across The Site
	SE3	Deleting Existing Cookie		STE2	High Number of Logins Across The Site
	SE4	Substituting Another User's Valid Session ID or Cookie		STE3	High Number of Same Transaction Across The Site
	SE5	Source Location Changes During Session	Reputation	RP1	Suspicious or Disallowed User Source Location
	SE6	Change of User Agent Mid Session		RP2	Suspicious External User Behavior
Access Control Exception	ACE1	Modifying URL Argument Within a GET for Direct Object Access Attempt		RP3	Suspicious Client-Side Behavior
	ACE2	Modifying Parameter Within A POST for Direct Object Access Attempt		RP4	Change to Environment Threat Level
	ACE3	Force Browsing Attempt			
	ACE4	Evading Presentation Access Control Through Custom POST			

Another categorization, which does not appear in the AppSensor documentation, has been created. This is partially to cater for the new "reputational" type, but also some of the new signature-based events. It divides the detection points into three classes:

- *Discrete*
Detection points that can be activated without any prior knowledge of the user's behavior and thus are related to the scope of the request
- *Aggregating*
Detection points that require a number of prior identical events to occur before they are activated and thus relate to activities over the duration of a single or multiple sessions (of one or more users)
- *Modifying*
Detection points that are typically only used to alter the detection thresholds or response actions

The detection points are shown categorized in this way in Appendix A, Table A-3.

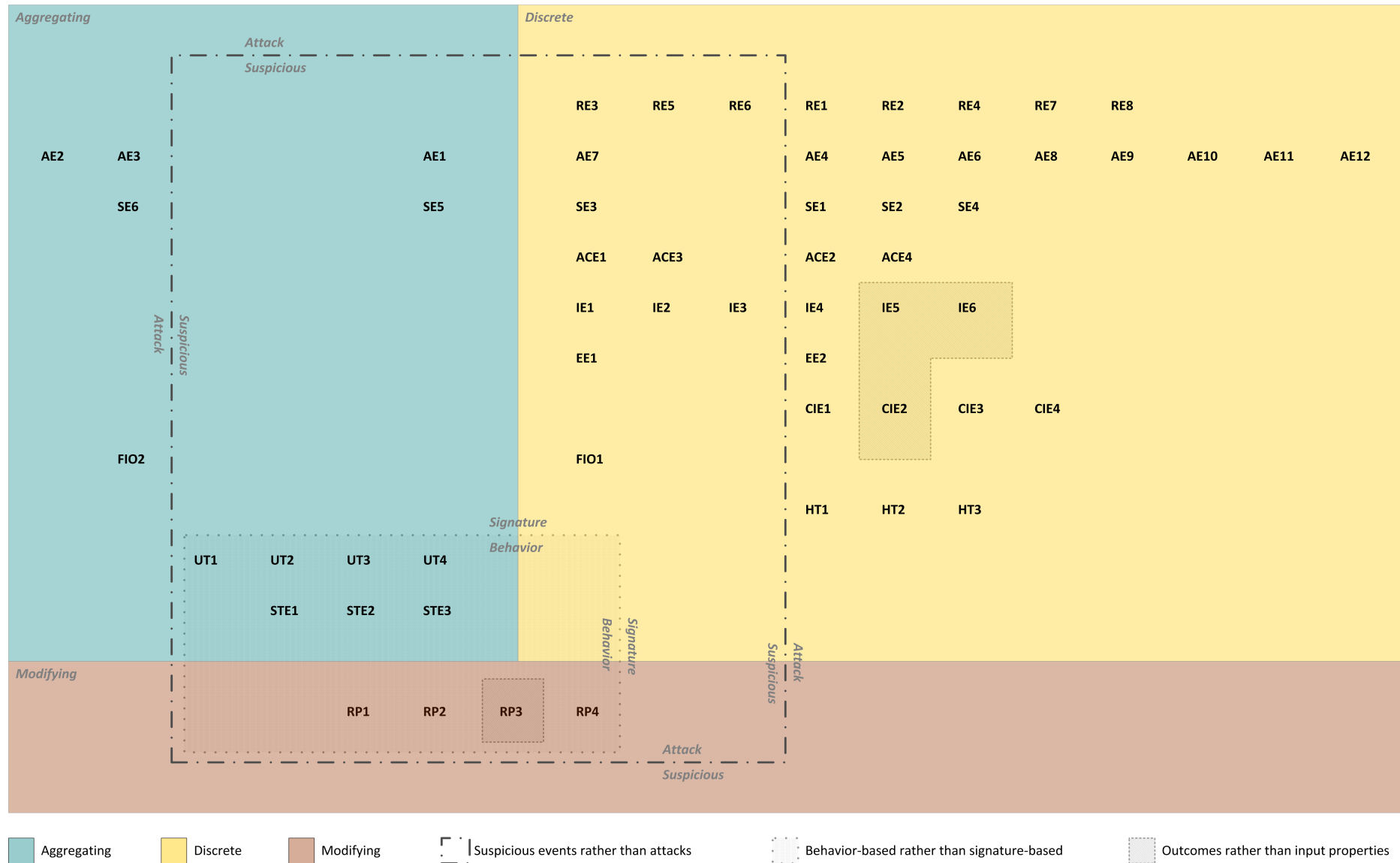
All three of these categorizations have been summarized in Figure 2-1 on the following page where:

- the detection points within each exception types run across the diagram horizontally, beginning with the Request Exceptions (RE) and finishing with the Reputation ones (RP) at the bottom of the diagram
- the detection point names and exception types can be found by reading the identity codes in Table 2-1 on the previous page
- discrete, aggregating and modifying detection points are separated and indicated by the colored areas
- suspicious events are bounded by the heavy dashed line
- behavior-based detection points are shown grouped together on the stippled background
- the four "outcome" detection points are indicated using a hatched background.

At a glance, we can now see that all behavior-based detection points are of the suspicious type, and all are of the aggregating class. The majority of the detection points are in the *discrete* class, and of those, most detect attack events.

Figure 2-1: Detection point categorization overview

Using information from the tables in Appendix A



2b) Requirements

Having defined the different detection point categorizations, the scope of the application must be defined. Each existing application should have documentation relating to its structure and functionality, and this information may be some of the artefacts produced during risk assessment. Where possible ensure the following are known:

- the different roles users fall into, and how these are allocated
- all the valid application entry points, whether POST and/or GET should be used and whether SSL/TLS is mandatory, optional or prohibited
- which of the entry points change state
- which users/roles have access to these entry points
- the broad functionality blocks and trust boundaries (e.g. data flow diagrams)
- the various inputs for each entry point (form, URL query string and path parameters, HTTP headers including cookies), and their data types and acceptable values
- which of the inputs may be manipulated by users and whether the interface for doing that is constrained (e.g. radio buttons and select elements) and whether there is any client-side validation for any of the elements
- whether there is functionality relating to authentication and session management.

Additionally, access to source code of an existing application can aid detection point selection and positioning, since there will be greater knowledge about data flow and security mechanisms that already exist.

Firstly we need to identify candidate detection points, then build these into a detection model. The candidate detection points can be selected using three main approaches:

- application risk classification
- threat assessment
- class categorization

These can also be used in combination.

Risk classification

A broad brush approach to select candidate detection points is to base it solely on the category types most appropriate for various application risk ratings. A suggested allocation is shown below in Table 2-2 which illustrates that "low risk" does not mean no instrumentation. Remember that risk is organization dependent, and the table below really needs to be adjusted for your own organization's risk tolerance and classification scheme.

Table 2-2: Applicability of AppSensor Detection Points to Risk Levels

Listed by category

CATEGORIZATION		APPLICATION RISK CLASSIFICATION			
SOURCE	TYPE/CLASS	LOW	MEDIUM	HIGH	CRITICAL
See 2a) on page 14	Inputs				
	Outcomes				
Table A-1	Signature				
	Behavior				
Table A-2	Suspicious				
	Attack				
Table A-3	Discrete				
	Aggregating				
	Modifying				

KEY

	Applicable
	Possible
	Not Applicable

Each organization could maintain its own lists of detection point requirements for the various risk classifications, and these would then form part of project requirements.

However, this type of approach is not recommended until a number of applications have been "instrumented" so that the organization has sufficient experience, and has been able to adjust the detection points to match its own risk needs.

Threat assessment

Instead of using a risk classification approach, the actual threats, possible vulnerabilities and the potential impacts can be used to select candidate detection points. In Appendix B there is a multi-part chart cross-referencing the detection points with two well-known classifications:

- Web Application Security Consortium (WASC) Threat Classification
<http://projects.webappsec.org/Threat-Classification>
 - Attacks
 - Weaknesses
- OWASP Top Ten 2010 - the ten Most Critical Web Application Security Risks
http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Therefore existing application threats assessments and other forms of risk analysis can be used together with these charts to identify candidate detection points. Also consider additional custom detection points for specific business logic threats that have been identified.

Class categorization

However, where possible the recommended approach is to consider the selection of candidate detection points based on the previously discussed three classes - *discrete*, *aggregating* and *modifying*.

Discrete

The *discrete* detection points are activated in the same time-frame of the request/response cycle. Note that normally multiple discrete events will be required before a certain threshold is reached to initiate an AppSensor response action.

For the *discrete* detection points, the approach builds on the suggestions made in the AppSensor documentation of differentiating between detection points that are usually best implemented in generic request pre-processing modules/filters (which are executed prior to normal page processing) and those that are best implemented in the business layer. Henceforth we shall use the following terminology for these two sub-classes of *discrete*:

- *Generic pre-processing*
- *Business layer*

The *discrete generic pre-processing* detection points should be implemented in the processing of requests before the *discrete business layer* ones. These are discussed in more detail below.

Generic pre-processing

The *generic pre-processing* sub-class used for broad request validation is independent of functionality, and should be used on each and every request received by the application. These might be coded using something like Java filters, ASP.NET HTTP handlers or an automatically prepended/included processing script or middleware hook in other languages and frameworks. The *generic pre-processing* detection points are trying to answer questions like:

- is this a well formed HTTP request?
- is this method allowed for the URL?
- is SSL/TLS being used when expected?
- is this user's session valid?
- is this a valid entry point (for this user/role)?
- are the required parameters provided and the parameter values of the correct general types?

The *generic pre-processing* detection points do not include more specific authorization and input validation checks (see *business layer*).

Figure 2-2 on the following page is based on the previous Figure 2-1, but additionally shows which detection points could be used in the *generic pre-processing* stage. The suitable detection point identity codes are highlighted in italicized underlined orange text. Note how these only encompass detection points in the discrete class—not any in aggregating or modifying classes (discussed later).

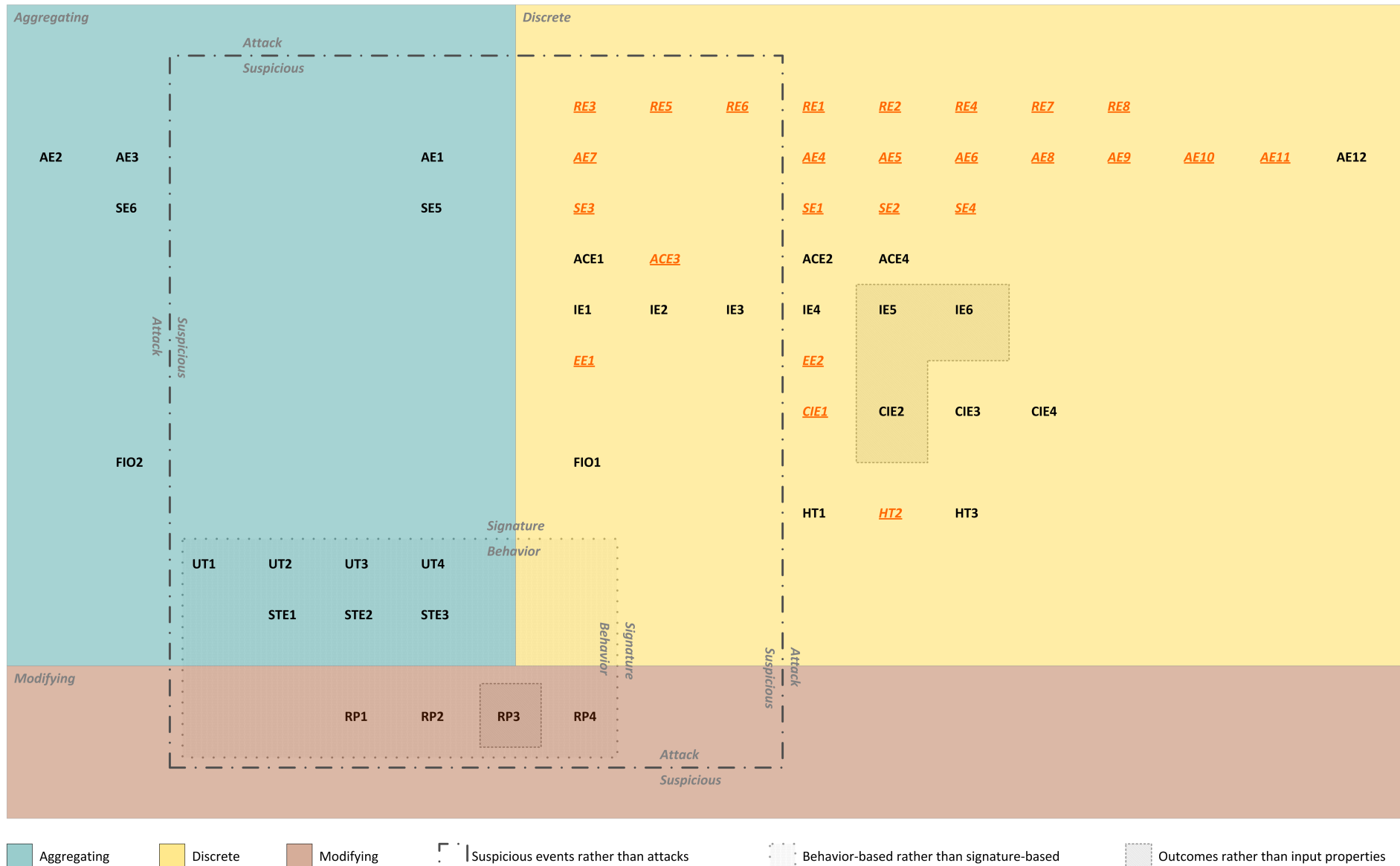
The information in Appendix A, Table A-3 (AppSensor Detection Points Categorized by Whether They are Discrete, Aggregating or Modifying) is reproduced in Appendix C, but with detection points in the aggregating or modifying categories grayed out. Table C-1 shows the detection point suitability for the *generic pre-processing* (broad request validation). This would be completed before any other detection points, so as many detection points should be included as are appropriate for the application. In practice this might include all all the detection points shown dark blue in Table C-1, and some aspects of the ones shown mid-blue:

- if there is no authentication or session management, exclude detection points AE8, 9, 10 & 11 and SE1, 2, 3 and 4
- corporate policy or lack of resources may mean it is not possible to implement the honey trap detection point HT2.

The reason for doing so many of the detection points at this stage, is that they can be centralized in the code. And since the approach will be similar for all applications, where possible this should be developed as a generic module or built into frameworks—the information on allowable URLs, parameters, methods, etc should be abstracted into some sort of data store that the detection points can reference.

Figure 2-2: Possible generic pre-processing detection points

Based on Figure 2-1 and using information from the Table A-4 in Appendix A; pre-processing detection points highlighted in italicized underlined orange



Business layer

Once the broad request validation has been undertaken, other detection points should be implemented in the *business layer*. These more detailed checks will depend upon the context, functionality, inputs and outputs and include aspects such as:

- input validation
e.g. checking against a whitelist; looking for XSS patterns;
- authorization
e.g. modification of parameter values for direct object access; force browsing
- authentication
e.g. log in, password reset, password change
- result inspection
e.g. number of records returned; data integrity

The selection of detection points depends on the functionality. In Appendix C Tables C-2 to C-4, we have presented suggested detection points for three aspects:

- Table C-2: Detailed authorization (e.g. validation beyond general permissions to script)
- Table C-3: Detailed request validation
- Table C-4: Detailed result inspection

Note that each request has more than one input aspect (e.g. each header, each parameter) and therefore multiple detection points of each code may be required. This is particularly true for the input exception type. Similarly result inspection may be implemented many times as different actions occur (e.g. database or xpath queries).

There are also suggested specific detection points for authentication functionality:

- Table C-5: Log in
- Table C-6: Password reset by an unauthenticated user
- Table C-7: Password change by an authenticated user

Other functionality (e.g. user input form) should already be covered by the detection points highlighted in Tables C-2 to C-4, but see also "Custom types" on the next page.

Aggregating

Examine the aggregating detection points and identify those that could be of use for the application's functionality. They may be

relevant to applications in any risk classification (see Table 2-2: Applicability of AppSensor Detection Points to Risk Levels). The signature-based aggregating detection points relate to authentication, session management and file input/output. If these are aspects of the application's functionality, include them in the candidate detection points.

The behavior-based aggregating detection points will be of particular benefit to higher risk applications; the system-wide detection points are especially important where there are a larger number of users or where the application's stability is a particular concern. Additionally where there is user-content sharing, the risk of application worms should be considered. The system trend detection points should be considered for these applications by examining each functional area and asking "what if usage changed by -50%, + 50%, +100%, +500%, etc". It will also be necessary to decide how to measure usage, if similar metrics are not already in use. For these, the sampling time period is important to—a 50% increase in a five-minute period is not nearly as significant as a 50% increase across a full day. But equally you may want to have to wait for a day to pass before finding out about this increase.

Modifying

The modifying detection points should be considered for higher risk applications (see Table 2-2: Applicability of AppSensor Detection Points to Risk Levels) where there is a perceived need for finer control over responses actions (see *step 3*).

An organization may have an existing source of reputational information it wishes to include (e.g. network device, a national or organization security threat level index). If this is the case, add relevant detection points to the list of candidate detection points.

Custom types

You may also want to create some custom business logic detection points that are particularly important for your own application. The second video presentation mentioned in the Introduction described some custom detection points Mozilla are using in a bookmark synchronization service for Firefox:

- credential mismatch in a URL
- tampering with a password reset code
- account deletion without a password.

Your own custom detection points may be derived during threat modelling exercises or from a knowledge of the applications specific functionality by examining use cases, mis-use cases and other requirements documentation. Where possible focus on developing signature-based detection points that indicate attacks (rather than just suspicious behavior).

Perhaps use the "CM" prefix for these.

2c) Model development

By this stage we now have a list of candidate detection points.

Now we need to define each candidate detection point in detail. The requirements should define:

- purpose
- general statement of its functionality
- details of any prerequisites
- related detection points.

The AppSensor detection point documentation (http://www.owasp.org/index.php/AppSensor_DetectionPoints) should be used as a starting point. In particular, read the examples and considerations for inspiration and things to watch out for. You may require multiple versions of the same detection point e.g. IE3 whitelist validation of parameter names, IE3 whitelist validation of IP addresses, etc.

For each point begin a specification sheet like the examples in Figure 2-3 and 2-4 on the following pages. These should identify the AppSensor identity code and the more specific purpose for the particular application.

The "Series" number in the figures will be used as the starting point numbering for sequential numbering of each detection point instance e.g. IE1-1001, IE1-1002, etc. It is possible to have identical AppSensor detection point identity codes (e.g. IE1) but with different purposes (e.g. the whitelist is source IP addresses rather than parameter values) and those should have a different series numbering e.g. 1000, 2000, etc.

At this stage, these specification sheets should be independent of where the detection points will be located, and should not include any consideration of response actions.

Aggregating detection points need slightly different specification. The trend and comparison period for each detection point must also be identified. For example these might include both technical and business tests:

- 5 different usernames tried in 30 minutes (AE1)
- the source location changes to any other continent (SE5)
- number of orders placed in 1 hour (UT1)
- number of logouts in 5 minutes (STE1)
- number of new site registrations in 15 minutes (STE3)
- number of shopping carts abandoned in 1 hour (STE3)

Figure 2-3: Example detection point definition overview for an instance of IE2

Initial requirements

DETECTION POINT DEFINITION - OVERVIEW			TYPE						
CODE/TITLE	IE2	Violation of Implemented White Lists							
SERIES/PURPOSE	3000	Detailed parameter validation against white list							
DESCRIPTION	<p>Whitelists are defined in XML data associated with the application for each allowed form and URL parameter. This detection point compares the parameter value with two whitelists:</p> <ul style="list-style-type: none">1) valid values: that can be used safely as inputs to subsequent processing2) invalid values: that should be rejected, but might only be user error (soft rejection) <p>Values that do not match either whitelist are invalid and impermissible (hard rejection).</p>								
PRE-REQUISITES	All generic pre-processing DPs								
RELATED DPs	None								
COMMENTS	<p>The parameters have been previously screened for missing/duplication/extra parameters and values. Some parameters can be defined but have NULL value.</p> <p>Some parameter values may be lists (e.g. comma delimited) of other values.</p>								
CHANGE LOG	<table><thead><tr><th>DATE</th><th>BY</th><th>ACTION</th></tr></thead><tbody><tr><td>19 Oct 2010</td><td>CW</td><td>Created</td></tr></tbody></table>			DATE	BY	ACTION	19 Oct 2010	CW	Created
DATE	BY	ACTION							
19 Oct 2010	CW	Created							

Figure 2-4: Example detection point definition overview for an instance of ACE3

Initial requirements

DETECTION POINT DEFINITION - OVERVIEW			TYPE												
CODE/TITLE	ACE3	Force Browsing Attempts													
SERIES/PURPOSE	1200	Validation of request URL against whitelist of allowable application surface													
DESCRIPTION	<p>All permissible application entry points are defined in the database, together with whether SSL/TLS is mandatory, optional or disallowed. The database also includes URLs of dynamic (e.g. scripts) and static (e.g. style sheets, images, etc) content entry points.</p> <p>This detection point is called for every HTTP request to the application.</p> <p>This detection point checks the path and whether SSL/TLS is being used.</p>														
PRE-REQUISITES	RE1, RE2														
RELATED DPs	RE3, RE4														
COMMENTS	<p>This DP does not validate user/role permissions for the URL or the presence/absence of parameters.</p>														
CHANGE LOG	<table><thead><tr><th>DATE</th><th>BY</th><th>ACTION</th></tr></thead><tbody><tr><td>19 Oct 2010</td><td>CW</td><td>Created</td></tr><tr><td>23 Oct 2010</td><td>AK</td><td>Note on exclusions added to comments</td></tr><tr><td>23 Oct 2010</td><td>MM</td><td>DP locations added</td></tr></tbody></table>			DATE	BY	ACTION	19 Oct 2010	CW	Created	23 Oct 2010	AK	Note on exclusions added to comments	23 Oct 2010	MM	DP locations added
DATE	BY	ACTION													
19 Oct 2010	CW	Created													
23 Oct 2010	AK	Note on exclusions added to comments													
23 Oct 2010	MM	DP locations added													

Once the specification sheets are complete, create a high-level overview of the application showing the main processing blocks/functionality perhaps in the style of a data flow diagram.

Then, using a list of the site's functionality and/or different usage scenarios together with the specification sheets, mark up the approximate positions of the various detection points identified. Many usage scenarios will have very similar data flows and can be grouped together.

Identify other systems the application exchanges data with and optionally include an indication of known trust boundaries. Examine the charts and look for additional detection point requirements. For example, consider input validation and the number of returned records (CIE2).

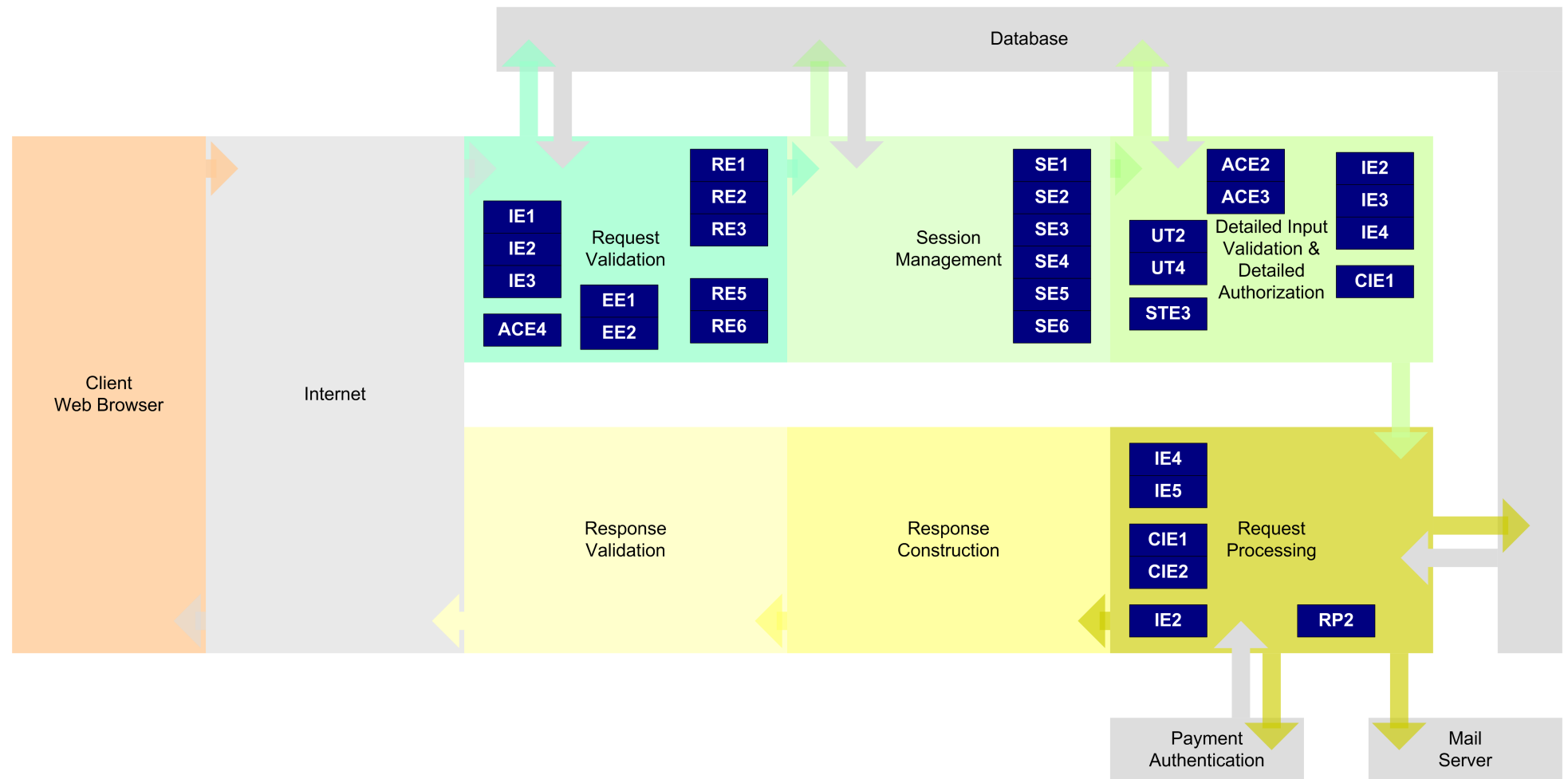
An example for the credit card authentication process in a payment application is shown in Figure 2-5 on the following page. This chart is reproduced in Appendix D together with additional examples.

These should begin to show how it makes sense to undertake the *discrete generic pre-processing* detection points in centralized functionality since it will be common to almost all requests. The *discrete business layer* detection points will be associated with particular application functions.

We now have our initial AppSensor model.

Figure 2-5: Example data flow diagram annotated with potential detection points

Initial requirements for one process



2d) Optimization

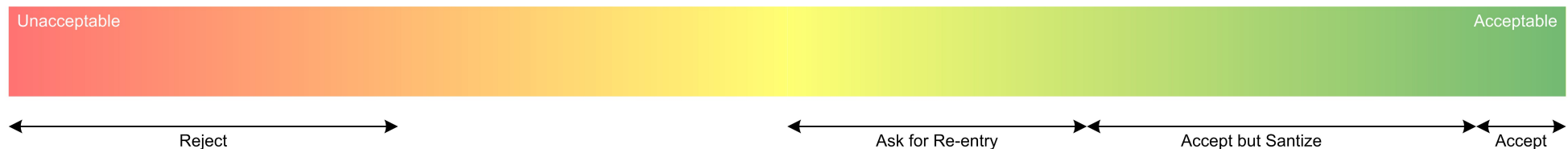
We now have a number of candidate detection points and have specified how they should be applied in request handling data flows. However, detection points can be specified in more than one way. We need to make sure the purposes and descriptions created perform correctly. Beginning with the specification sheets and data flow diagrams, we now want to optimize our detection point model in three ways:

- to maintain a low false positive rate through adjusting the sensitivity
- to consider relationships with other systems and the effects these may have on detection points
- to determine if any detection points can be removed to eliminate overlaps and duplicates.

Sensitivity tuning

During this stage, consider what could go wrong with input data. We must ensure that the detection points are tuned to detect malicious behavior and not just user errors—some could be specified in a way that leads to more false positives. In Figure 0-1, the range of user behavior was used to illustrate that malicious attacks are separate to normal application use. Figure 2-6 shows how this approach can be applied to individual input values where the type and format of an acceptable value may have some tolerance between what is acceptable and what is unacceptable:

Figure 2-6: Normal application use means validation needs to cater for a range of acceptable user data values



Some "invalid" user data examples are shown in Figure 2-7 on the following page. Users may copy and paste information into form fields, or put the data in the wrong field, or use an unexpected format such as when entering a phone number. Applications should allow some degree of variation in user behavior and thus allow for normal user error.

We must check that our proposed detection points will not inadvertently flag what might be normal behavior as an attack. For each detection point, examine possible scenarios where the detection point might be fired by normal, or non-malicious use. This will help tune the system helping us choose appropriate response actions (later). For each detection point consider:

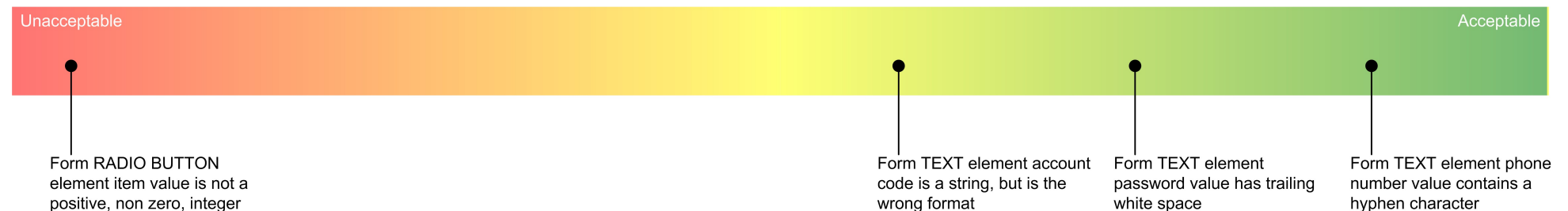
- automated non-malicious systems (e.g. web crawlers)
- human error (misunderstanding, typographical)
- input device errors (e.g. conversion of voice to text, truncation of a URL in a link sent by email)
- specificity of error threshold (e.g. space, hyphen and parentheses characters in a telephone number, past/future application changes (e.g. old URLs, changes to forms)
- network configuration and architecture.

For example, an application's entry points are well defined and a detection point is chosen to be activated when a request is made for any other URL (e.g. force browsing, URL whitelisting). The application may be able to monitor 404 errors and other invalid URLs using an internal module or it could consume such data from another device (e.g. web server logs or a web application firewall) if this can be done in real time. But in a public application, you are likely to receive a large number of non-malicious 404s and these will not normally be attacks. The ability for AppSensor to maintain a low false positive rate in this example this depends upon the way the detection point and response are specified.

Another example would be an invalid ID parameter. If the options are provided to the user in a constrained interface element like a form select element, it is more suspicious than if there are some unexpected characters in a form text element.

Figure 2-7: Some data unacceptable data values will be much more likely to indicate a malicious user

Acceptability is organization and application dependent



Some examples for detection points which could be susceptible to these types of sensitivity problems are expanded upon in Appendix E. Consider these in the context of the application and the way in which the input aspect (URL, headers, parameter name or value) might conceivably be provided by the user.

Effects of related systems

The actual context is also important. If a data entry form has some presentation-layer (client-side) validation in addition to equivalent matching server-side validation, and the submitted data includes problems which the presentation-layer validation should have caught, the acceptability of the inputs may be different. If we assume type and format and lengthy validation on the client side, Figure 2-7 changes considerably.

Figure 2-8: Application-layer knowledge increases the ability the ability to differentiate between normal and malicious input



Similarly, if a request or data are received from a trusted information system, the standard of tests to validate the data could be stricter. XML data which has been validated by an XML Firewall should be of higher quality, and less prone to human errors, than that in an RSS feed pulled directly from another website. Do not trust either source completely, but consider the seriousness of a detection point being activated from a more reliable source.

Therefore consider the original source of data being processed. Was it user-generated content, was it retrieved from a reliable source, and if the latter do we know what verification has already been performed? This analysis may lead to the creation of additional detection point instances of the same detection point identity code, but they have different requirements and are used on different types of input.

Model reduction

Finally, we want to remove any duplication of effort - using the same detection point more than once on the same input or using another detection point which does not add any further value.

This process is undertaken by examining the model to check that detection points with the same functionality are not being repeatedly called on the same data. Note that the same detection points may correctly occur many times within the processing of a request such as when each parameter value is checked against a whitelist.

It is also possible that the way some detection points have been specified in a manner which negates the need for others. Check whether a very specific detection point is already tested in a less specific detection point. For example if AE10 (adding additional POST variables) is proposed for the application's authentication module and broad request validation includes RE5 (additional/duplicated data in request) it may be possible that AE10 is not adding any further detection. Provided these are given identical priority, there is no need for both, or the RE5 could be modified to capture the functional area or purpose, which might then be used to affect the response action. But note it may still be useful to record that the action was the more specific AE10 (as well as RE5), and another option would be to alter the specification for RE5 so it can activate AE10 type events at the same time, if it knows it is an authentication request.

Figure 2-9 below uses link arrows to show possible inter-relationships between detection points. Depending upon how the detection points have been specified, the source of a link arrow might be a more generic version of the destination of the link arrow. This does not mean the source necessarily caters for all possibilities, but can be useful in avoiding duplication. But check that removing a detection point does not mean that an aspect is left uncovered in another attack.

Update the specifications and charts with any changes required.

Now create test cases for requests that should activate the detection points. Try to create separate tests for each detection point, and this may mean hundreds of test cases since they will include at least one for every parameter submitted in requests.

Lastly, review application design/functionality that changes the flow through code and especially any blocking actions (e.g. redirects, session termination, custom error page display). Check whether any of these circumvent or prevent detection points from being activated. For example the application might already lock an account for 20 minutes after three invalid passwords are provided in a 24hr period but AE2 (multiple failed passwords) might be specified requiring a different number.

Figure 2-9: AppSensor Detection Points Inter-Relationships

2e) Code location

Now that the particular detection point requirements have been completed, it may be necessary to define specifically where in the code the detection points will be added. The previous specifications and charts should help identify common detection points and with a knowledge of the application's code or proposed architecture, the detection point locations can be defined.

Create a record of the proposed detection point locations such as the partial examples in Figures 2-10 and 2-11. Whitelist input validation (a *discrete business layer* detection point) may occur in very many locations in the application code. *Discrete generic pre-processing* detection points are likely to exist in very much fewer, and possibly a single, locations.

Remember the detection code should already be there in an existing securely coded application; we are simply adding the instrumentation to collect that information together, and act on it.

Figure 2-10: Example partial detection point schedule for IE2

Partial view of a longer schedule

DETECTION POINT DEFINITION - SCHEDULE					TYPE	
CODE/TITLE		IE2 Violation of Implemented White Lists				II - Discrete / Business layer
SERIES/PURPOSE		3000 Detailed parameter validation against white list				
LOCATIONS	ID	OBJECT	MODULE	FUNCTION	ENTRY POINTS	NOTES
	IE2-3010	username	site.dao.auth	checkUser	/	
					/login.jsp	
					/reset.jsp	
	IE2-3011	password	site.dao.auth	checkCredentials	/	
					/login.jsp	
	IE2-3012	resource	site.dao.auth	checkResource	/	
					/login.jsp	
	IE2-3020	press_release	site.dao.media		/media/pressrelease.jsp	

Figure 2-11: Example detection point schedule for ACE3

DETECTION POINT DEFINITION - SCHEDULE					TYPE	
CODE/TITLE	ACE3	Force Browsing Attempts			I - Discrete / generic pre-processing	
SERIES/PURPOSE	1200	Validation of request URL against whitelist of allowable application surface				
LOCATIONS	ID	OBJECT	MODULE	FUNCTION	ENTRY POINTS	NOTES
	ACE3-1210	URL	site.dao.request	checkURL	[All]	

2f) Attack analysis

The last stage (of *step 2*) for detection point selection is to undertake an attack analysis. Select attacks that have been identified from threat assessments, or if this is not available consider those shown in Appendix C from:

- WASC Threat Classification v2.0
<http://projects.webappsec.org/Threat-Classification>
- OWASP Top Ten 2010
http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

A full catalog of attack patterns for all types of software is provided in CAPEC:

- Common Attack Pattern Enumeration and Classification (CAPEC)
<http://capec.mitre.org/>

Use both likely attacks identified during risk assessments as well as feasible but much less likely attacks. Remember, we are concerned with identifying and stopping attacks against unknown vulnerabilities such as:

- a SQL injection point introduced during a change to the application which was missed due to insufficient testing
- a zero day vulnerability in a code library used by the application

For each attack, consider a range of valid and invalid application entry points, and check the model through using the real attacks. Examine all the detection points which might be activated, ignoring for the moment what their response may be. List all the detection points for each attack scenario and determine whether these are reasonable, and provide sufficient coverage. Then consider if it is possible for human errors to generate the same situation. If so, reassess the detection points proposed.

If necessary, re-iterate through stages 2c to 2f to finalise the selection of detection points.

We should now have the following artefacts:

- detection point specifications
- schedule of detection point locations
- test cases

and can proceed to response actions in *step 3*.

3. Response action selection

- a) Strategic requirements
- b) Thresholds
- c) Model tuning



In AppSensor a response is a change in application behavior; it is not any form of retaliation. The response aims to defend the application, its users and everyone's data:

- organization data
- user data (sometimes including PII/personal data)
- data belonging to other parties (e.g. suppliers, customers and partners)

Table 3-1 categorizes the fourteen AppSensor response actions by their effect on the user, i.e. from the user's viewpoint. It also shows the broad purposes, whether the target of the response affects a single user or all users and the duration of the action.

Table 3-1: AppSensor Responses

Listed by category, then alphabetically by ID

CATEGORY		RESPONSE		CLASSIFICATIONS				TARGET USER		RESPONSE DURATION		
TYPE	DESCRIPTION	ID	DESCRIPTION	LOGGING	NOTIFYING	DISRUPTING	BLOCKING	ONE	ALL	INSTANTANEOUS	PERIOD	PERMANENT
Silent	User unaware of application's response	ASR-A	Logging Change	●				●	○	○	○	
		ASR-B	Administrator Notification	●	●			●	●	●		
		ASR-C	Other Notification	●	●			●		●		
		ASR-N	Proxy	●				●	○		●	○
Passive	Changes to user experience but nothing denied	ASR-D	User Status Change	●				●			●	
		ASR-E	User Notification	●	●	●		●		●		
		ASR-F	Timing Change	●		●		●	○	○	○	
Active	Application functionality reduced for user(s)	ASR-G	Process Terminated	●	○	●		●		●		
		ASR-H	Function Amended	●	○	●	●	●	○		●	○
		ASR-I	Function Disabled	●	○	●	●	●	○		●	○
		ASR-J	Account Logout	●	○	●	●	●		●		
		ASR-K	Account Lockout	●	○	●	●	●			●	○
		ASR-L	Application Disabled	●	○	●	●		●			●
Intrusive	User's environment altered	ASR-M	Collect Data from User	●				●			●	

Source: http://www.owasp.org/index.php/AppSensor_ResponseActions

ASR-P for "no response" (not shown in Table 3-1) is usually only output in logs to indicate an event did not initiate a response.

3a) Strategic requirements

We need to consider the organization's risk tolerance and the desired user experience (e.g. acceptability of changes to service level and function availability, reduced usability). Remember "users" are not always people and can be other information systems. The selected response actions will also depend on the purpose of the application such as whether it is a sales channel, a marketing asset, a service for citizens, a mission critical process or safety critical system. Here are some types of view which different organizations may have:

- Only allow a few security events that are obviously attacks or several minor events which are just suspicious.
- Do not prevent users doing anything, but log, monitor and alert fervently.
- Never log out or lock out site administrators, but ensure they are aware of all suspicious and attack events, and know that their own activity is being recorded in tamper-evident audit logs with any AppSensor alerts being sent to their supervisors.
- Any two events each with more than 75% certainty that it is an attack must log the user out and lock their account immediately, and this can only be reset by two administrators from different locations acting together.
- Never disable any functionality.
- Authenticated administrators who have access to the most functionality and the greatest data access permissions should have the strictest thresholds before a response action is undertaken.
- Active (against the user) responses will only be used for users external to the corporate network.
- Active responses will only be used for users internal to the corporate network.
- Application functionality will not be changed unless the user's source location is in a higher-risk country.
- Ensure the user is oblivious to the response actions being taken.
- Nothing must be done which might affect WCAG 2.0 Level AA Conformance.
- Public unauthenticated users are the least trusted and should have the most strict thresholds.

Use these example rules to begin a workshop-style discussion and define some high-level policies before attempting to formalise the planned responses (thresholds and actions). Consider working through the main application entry points or functionality and, from the perspective of each user role, write some general rules for response that are allowed and appropriate. Take into consideration the effect the response actions might have on users and other systems, as well as the particular application. There should be less focus on technical issues, and more from user experience and business risk viewpoints. We are interested in the why, not the how. The facilitator should be able to steer the group so that all the aspects are covered. Then try to reduce these to 5-10 rules that apply to all users and all detection points. But you can see from the above examples, it is likely there will be demands for greater granularity in the response actions, and you may want to allow for this in your frameworks, code libraries and outsourced software specifications.

3b) Thresholds

From above, we should have a small number of high-level rules to guide threshold selection. Initially we will exclude the consideration of detection points in the modifying class, since these are normally used to adjust default thresholds and actions. Thresholds need to be set for how many security events are allowed to be created before a response is made.

It is important to separate the application's own responses from those of AppSensor. An application may lock accounts due to multiple failed authentication attempts or it might block requests using a disallowed HTTP method. But AppSensor still needs to record and monitor these to undertake responses in addition to the application's normal behavior.

Two approaches need to be considered:

- whether the responses are dependent upon user role (e.g. authenticated versus unauthenticated)
- whether responses are set on a per detection point basis, or a per application basis.

The high-level rules should provide guidance on the first of these. If AppSensor is only implemented for the authenticated part of an application, or there is only one role, this question needs no further consideration. Applying different thresholds to different roles does create additional complexity, and some detection points and responses may not be valid for certain roles (e.g. authentication and session management exception types).

Response threshold definition based on a per detection point basis allows more fine-grained tuning. However it is usual to have thresholds for each detection point as well as an overall limit on the total number of any detection points activated in a time period. The time period over which each threshold applies needs to be long enough to cater for slow attacks, but will need to be selected with consideration of any active responses that have time factors such as lockout for a period. Having the overall limit can help allow the individual thresholds to be much more tightly set.

User events and user trends

User-based events relate to detection points being activated by a single user. The definition of what constitutes a single security event is shown in Table 3-2.

Table 3-2: Weightings of suspicious and attack events

Grouped by whether they apply to one user's activity, or potentially all users

EVENT TYPE	NUMBER OF EVENTS EQUIVALENT TO ONE SECURITY EVENT
Suspicious	3
Attack	1

Source: Recommended Thresholds, AppSensor Project v1.1, https://www.owasp.org/images/2/2f/OWASP_AppSensor_Beta_1.1.pdf

Each detection point will have its own threshold of a small number of security events before a response action is taken. Then also consider the total number of security events generated by all detection points—the latter should normally all be set with the same period e.g. one day. Sample individual and overall thresholds are shown in Tables 3-3 and 3-4 below.

Table 3-3: Example response action thresholds for individual detection points

The threshold is the number of security events in the defined period per user

DETECTION POINT	THRESHOLD	PERIOD	ACTIONS	RESPONSE CODES
RE1	2	1 hour	Request terminated + Account lockout 30 minutes	ASR-G, ASR-K
RE6	10	5 minutes	Security violation message + Account logout	ASR-E, ASR-J
CIE1	3	15 minutes	Security violation message + Function disabled	ASR-E, ASR-I
HT3	1	NA	Admin alert + Proxy to alternative system	ASR-B, ASR-N

Table 3-4: Example response action thresholds for the overall number of security events

The threshold is the number of security events in the defined period per user

DETECTION POINT	THRESHOLD	PERIOD	ACTIONS	RESPONSE CODES
(All)	5	24 hours	Security violation message	ASR-E
(All)	30	24 hours	Security violation message + Account logout	ASR-E, ASR-J
(All)	45	24 hours	Security violation message + Account lockout 5 minutes	ASR-E, ASR-K
(All)	60	24 hours	Security violation message + Account lockout 30 minutes	ASR-E, ASR-K
(All)	100	24 hours	Security violation message + Account lockout indefinite	ASR-E, ASR-K

Source: Recommended Thresholds, AppSensor Project v1.1, https://www.owasp.org/images/2/2f/OWASP_AppSensor_Beta_1.1.pdf updated with response action codes and different thresholds

A threshold of "1" or a percentage comparison, such as shown for the HT3 detection point in Table 3-3, means the threshold is reached immediately, and no time period needs to be defined. The longer the period, the more strict is the policy.

Consideration also needs to be given to situations where multiple detection points are activated with a single request. This is not unlikely and two examples are:

- a SQL injection attack leads is detected as a Command Injection exception (CIE1), but also fails Input Exception whitelist checks (IE2) and Request Exception due to other missing parameters (RE6)
- separate Input Exception validation checks may identify problems with different parameters (e.g. IE2, IE2, IE2, IE3, IE4, IE4).

In these cases, one request could lead to an individual detection threshold being exceeded more rapidly than expected or even the overall threshold being reached very quickly. It is important to record every event, but for some applications it may be necessary to

limit the contribution to the overall threshold as only one security event per request/response cycle. if possible, make this a configuration setting.

In a more advanced implementation may be able to track the exact event details, so that duplicate suspicious security events are not necessarily counted twice. For example, if a user submits an authentication form with the same wrong password twice, that doesn't usually provide twice as much evidence of an attack i.e. if AE5 (Unexpected Quantity of Characters in Password) is activated twice with the same value, this may be less significant than two AE5 activations by the same user but with different values.

The weightings in Table 3-2 could also be altered for each detection point, rather than just on suspicious versus attack, but the recommendation is not to alter these weightings and instead alter thresholds (number and period) only.

Security event logs may include a confidence rating, defining how certain the event identification is. In AppSensor, the detection points should have been selected and their sensitivity tuned so that the confidence is very near 100% all the time. In other words, weighting based on confidence should not be required.

Different thresholds and response actions based on the application's risk classification rating as shown in Table 3-5.

Table 3-5: Example response action thresholds based on risk classification

NUMBER OF SECURITY EVENTS FOR RISK CLASSIFICATION				ACTIONS	RESPONSE CODES
LOW	MEDIUM	HIGH	CRITICAL		
5	2	2	1	Security violation message	ASR-E
10	3	3	2	Security violation message + Account logout	ASR-E, ASR-J
NA	5	5	NA	Security violation message + Account lockout 5 minutes	ASR-E, ASR-K
20	10	7	3	Security violation message + Account lockout 30 minutes	ASR-E, ASR-K
50	25	10	NA	Security violation message + Account lockout indefinite	ASR-E, ASR-K

From the examples in all three tables above, you can see that many responses are combinations of actions. Events will always be logged, but many of the actions will be combined with user, administrator or other forms of notification.

System trends

It is difficult to provide general guidance on system trend response actions. But having an automated response to a sudden shift in system activity is one of the benefits of using AppSensor.

The thresholds to initiate a response action need to be considered once the range of normal behavior has been examined over a period of time. This also needs to consider special situations that could alter the normal patterns of usage such as vacations, time of day,

newsworthy events and marketing activities, so that benign but variable site usage is not flagged as an attack. Therefore thresholds would usually include administrator notification levels before disabling a particular feature of the whole site. The existing AppSensor documentation provides a good example of this:

Table 3-6: Example system trend response action thresholds

Detection point monitoring the usage rate of an application's "Add a Friend" feature

SYSTEM TREND DELTA	ACTIONS	RESPONSE CODES
+1000% (5 minutes)	Administrator notification	ASR-B
+200% (15 minutes)	Administrator notification	ASR-B
+200% (60 minutes)	Administrator notification	ASR-B
+500% (15 minutes)	Administrator notification	ASR-B
+1000% (15 minutes)	Temporarily disable Add a Friend feature	ASR-I

System trend events should not be included in the overall (user) threshold mentioned above. By their nature they are very specific and will rarely add anything to knowledge about an individual user. Similarly there is no need for an overall system trend threshold.

Modifying

The reputational detection points (RP1-4) can be used to dynamically alter thresholds in real time. For example if an organization tracks the national terror threat level and such aspects are considered threats to the application, the thresholds could alter in response to this (RP4). However, the degree of trust in the source, availability and accuracy of information needs to be considered with each detection point. Some (like the national terror threat example) would require a threshold of "1" if the intention is to make a change in AppSensor's response as soon as the event occurs.

Any change which disables a user, feature or the whole application could be used to perform a denial-of-service attack, and therefore responses to activation of detection points in the modifying class should be chosen conservatively.

Overall summary

For all thresholds, define whether counts are ever reset, e.g. at the end of a session, when an application is restarted.

Figure 3-1 on the following page shows part of an example schedule documenting the application's threshold settings. This shows that some of the session management exceptions only have meaning for a period that equals the session length, and that some aggregating detection points will have thresholds of "1" where they act like an off/on switch.

Figure 3-1: Example schedule of thresholds

Partial view of a longer schedule that has many more user event definitions

RESPONSE ACTIONS - SCHEDULE OF THRESHOLDS

USER EVENTS (INDIVIDUAL DETECTION POINTS)

CODE	SERIES	THRESHOLD	PERIOD	RESPONSES
RE1	1000	2	1 hour	ASR-G
RE2	1000	2	1 day	ASR-G
RE3	1000	5	1 day	ASR-B, ASR-J
RE4	1000	5	1 day	ASR-B, ASR-J
AE2	1000	1	NA	ASR-K
AE3	1000	1	NA	ASR-K
SE1	1000	1	(session)	ASR-J, ASR-B, ASR-E
SE2	1000	1	1 day	ASR-A
SE5	1010	1	(session)	ASR-A
SE5	1020	1	(session)	ASR-B, ASR-K
ACE1	1000	2	30 days	ASR-B, ASR-K
ACE2	1000	2	30 days	ASR-B, ASR-K
ACE3	1000	5	15 minutes	ASR-A, ASR-F
IE1	1000	2	1 day	ASR-A, ASR-E, ASR-G
IE2	1000	1	1 day	ASR-G, ASR-B
IE2	1010	25	2 hours	ASR-B, ASR-J

USER TRENDS (INDIVIDUAL DETECTION POINTS)

CODE	SERIES	THRESHOLD	PERIOD	RESPONSES
UT1	1000	10	1 hour	ASR-B
UT1	2010	5	15 minutes	ASR-B, ASR-E
UT1	2020	40	1 day	ASR-E, ASR-B, ASR-I
UT3	1000	1	NA	ASR-D
UT3	2000	1	NA	ASR-B, ASR-I

OVERALL NUMBER OF SECURITY EVENTS

CODE	SERIES	THRESHOLD	PERIOD	RESPONSES
(All)	-	5	1 day	ASR-E
(All)	-	45	1 day	ASR-E, ASR-J, ASR-K

SYSTEM TRENDS (INDIVIDUAL DETECTION POINTS)

CODE	SERIES	THRESHOLD	PERIOD	RESPONSES
STE1	1000	+500%	NA	ASR-B
STE1	1000	+1000%	NA	ASR-B

3c) Model tuning

Once the thresholds and actions have been determined, final tuning of the model should be undertaken to ensure that the combined model behaves as required.

Method

Tuning is usually best accomplished by facilitating a discussion which includes members from various parties concerned with the application.

1. For each of the attacks defined in *step 2f* and examine whether the responses are as desired.
2. Examine typical user activities and introduce all types of input which could be accidental to check how much tolerance there is for:
 - misunderstandings
 - typing errors
 - copying and pasting formatted text
 - navigation changes such as using bookmarks, partial links or the back and forward buttons.
3. Consider slow and fast use of the application, and how often each function might be requested.
4. Consider the response to static content (e.g. RSS feeds, style sheets, video, images, JavaScript files, HTML files) requests.
5. Consider requests for missing content.
6. Examine carefully activities that can lead to active responses that disable part or all the application.
7. How do the range of available responses affect the wider system and related systems (interdependencies and interoperability)?
8. Consider the effect of the planned responses on other metrics such as uptime of the application and other systems, application response times, user satisfaction, throughput requirements and other business measures.

Some organizations may be able to use information from usability testing studies to assist with the second item. For example, disabling the whole application could stop further recording of security events and even prevent an administrator from re-enabling the application if that function is usually undertaken using a web interface which is part of the application.

Modify the detection points, response thresholds and actions if necessary.

Other considerations

Some broader issues to consider when developing the implementation plan are:

- should there be an option to overrule all responses so that they log only?
- could this "log only" option for certain source locations (e.g. an IP address) which is of limited time duration, raises administrative alerts when set, removed or expires and includes a process for management approval?
- can AppSensor data be exported into vulnerability management programs.
- can AppSensor data be exported in real time to security integration manager (SIM) systems?

And, in terms of development and release:

- use secure coding practices for AppSensor implementation.
- ensure AppSensor's correct implementation will be verified (the correct detection points are activated and the correct response action occur) through the use of testing processes at launch and periodically thereafter:
 - using attacks in 2f
 - using the test cases developed in 2d
 - using security and usability testing
- ensure existing change control processes are utilized for deployment.
- build in time to allow tuning of the system, especially to configure response thresholds.

Once these have been discussed and decisions made, the model is ready for implementation.

Conclusion

We have seen how AppSensor allows real-time detection and response to be built in to application. Whenever possible, AppSensor requirements should be built into project requirements from an early stage. For existing applications, the additional coding must be subject to the same secure development process as another other software changes. This includes risk analysis, design and code review and testing, operational enablement, etc.

We have seen how candidate detection points can be selected based on the application risk classification, or from threat modelling or from an assessment of functionality based on three classes of detection points. The detection points can then be specified in more detail to provide an initial model. This needs to be optimized and then consideration can be made of appropriate response thresholds and actions to produce a final model for implementation.

The process leads to the creation of specifications, test cases and schedules of detection points which can be used in procurement documentation and as reference materials during development processes.

Additional code adds complexity. However if an application has already been developed with security built in, the locations for detection points are likely to already exist and some local actions may already be being used (e.g. reject the input, ask the user to re-enter text, log the user out, etc).

A more rapid approach is described on the following pages.

Quickstart implementation plan

Organizations wishing to pilot AppSensor, or who want to use a fast-track approach can use this simplified baseline guide.

Table 2-2 (Applicability of AppSensor Detection Points to Risk Levels) on page 19 has been reproduced below with an extra column labelled "Baseline" to show that a lightweight implementation plan should focus only on input, signature based detection points that detect attacks rather than suspicious events in the discrete class.

Table 5-1: Baseline AppSensor Detection Points compared to Risk Levels (Table 2-2)

Listed by category, then alphabetically by ID

CATEGORIZATION		APPLICATION RISK CLASSIFICATION				BASELINE	KEY						
SOURCE	ITEM	LOW	MEDIUM	HIGH	CRITICAL								
See 2a) on page 14	Inputs						<table><tr><td></td><td>Applicable</td></tr><tr><td></td><td>Possible</td></tr><tr><td></td><td>Not Applicable</td></tr></table>		Applicable		Possible		Not Applicable
		Applicable											
	Possible												
	Not Applicable												
Outcomes													
Table A-1	Signature												
	Behavior												
Table A-2	Suspicious												
	Attack												
Table A-3	Discrete												
	Aggregating												
	Modifying												

This reduces the number of detection points to be considered. The suggestions for inclusion in a baseline implementation are

- Request exceptions (RE1, 2, 3 and 4)
- Access control exception (ACE1 and 2)
- Input exceptions (IE1, 2 and 3)
- Authentication exceptions (AE1, 2 and 3) when authentication is used
- Session Management exceptions (SE5 and 6) when session management is used.

In a lightweight implementation possibly limit the response actions to:

- additional logging (ASR-A)
- administrator notification (ASR-B)
- account logout (ASR-J)
- account lockout (ASR-K).

Initially use a fixed set of thresholds for all per detection point thresholds, but build in the capability to change these.

This approach can be implemented with your own code. But it can also be implemented using the AppSensor code which has been produced for ESAPI. This can be used independently of the rest of ESAPI, but the AppSensor code does rely on some ESAPI methods, but these could be reproduced in some other way. The core elements of AppSensor are available as a Java jar, ready to use with or without ESAPI and there is a developer guide available (http://www.owasp.org/index.php/AppSensor_Developer_Guide). The AppSensor jar requires three lines of configuration in the ESAPI.properties file and policies are defined in the appsensor.properties file. Then each detection point can be added directly in the code where appropriate (2-3 lines each). There is even demonstration code available in a tutorial lesson-based application (<http://code.google.com/p/appsensor/source/browse/trunk/AppSensor-Tutorial/>).

Finally, if full knowledge is available about the application's entry points (aka "attack surface"), the lightweight implementation should be extended to enforce this using:

- Request exceptions (RE5 and RE6)
- Access control exception (ACE3).

This will provide a significant filter for many probes by attackers.

Monitor the effects of AppSensor and determine whether it can be extended or applied to other applications.

List of tables

Number: Title	Page
2-1: AppSensor Detection Points Listed by exception type, then alphabetically by ID from http://www.owasp.org/index.php/AppSensor_DetectionPoints	17
2-2: Applicability of AppSensor Detection Points to Risk Levels Listed by category	21
3-1: AppSensor Responses Listed by category, then alphabetically by ID	41
3-2: Weightings of suspicious and attack events Grouped by whether they apply to one user's activity,, or potentially all users	43
3-3: Example response action thresholds for individual detection points The threshold is the number of security events in the defined period per user	44
3-4: Example response action thresholds for the overall number of security events The threshold is the number of security events in the defined period per user	44
3-5: Example response action thresholds based on risk classification	45
3-6: Example system trend response action thresholds Detection point monitoring the usage rate of an application's "Add a Friend" feature	46
5-1: Baseline AppSensor Detection Points compared to Risk Levels (Table 2-2) Listed by category, then alphabetically by ID	51
A-1: AppSensor Detection Points Categorized by Signature and Behavior Based Events Grouped by whether they apply to one user's activity,, or potentially all users	59

A-2: AppSensor Detection Points Categorised by Suspicious and Attack Events	60
Grouped by whether the detection points apply to one user's activity, or potentially all users	
A-3: AppSensor Detection Points Categorised by Whether They are Discrete, Aggregating or Modifying	61
Grouped by whether they apply to one user's activity, or potentially all users	
A-4: AppSensor Detection Points Categorised by Generic Pre-Processing vs Business Layer Implementation	62
Grouped by whether they apply to one user's activity, or potentially all users	
B-1: WASC Threat Classification attacks cross-reference with detection points (part 1/2)	64
Request, authentication, session and access control exceptions	
B-2: WASC Threat Classification attacks cross-reference with detection points (part 2/2)	65
Input, encoding, command injection, file input/output, honey trap, user trend and system trend exceptions, and reputation	
B-3: WASC Threat Classification weaknesses and OWASP Top Ten Risks cross-reference with detection points (part 1/2)	66
Request, authentication, session and access control exceptions	
B-4: WASC Threat Classification weaknesses and OWASP Top Ten Risks cross-reference with detection points (part 2/2)	67
Input, encoding, command injection, file input/output, honey trap, user trend and system trend exceptions, and reputation	
C-1: Broad request checks	69
Relevant to all entry points	
C-2: Detailed authorisation (e.g. validation beyond general permissions to script)	70
Possible detection points in addition to those in Table C-1 for broad request checks, relevant to all entry points	
C-3: Detailed request validation	71
Possible detection points in addition to those in Tables C-1 and C-2 for broad request checks and detailed authorisation checks	
C-4: Detailed result inspection	72
Possible detection points in addition to those in Tables C-1, C-2 and C-3	
C-5: Log in	73
Possible authentication-specific detection points in addition to those in Tables C-1, C-2 and C-3	
C-6: Password reset by an unauthenticated user	74
Possible authentication-specific detection points in addition to those in Tables C-1, C-2 and C-3	

C-7: Password change by an authenticated user	75
Possible authentication-specific detection points in addition to those in Tables C-1, C-2 and C-3	
E-1: Detection point tuning analysis considerations (part 1/4)	80
Request exceptions	
E-2: Detection point tuning analysis considerations (part 2/4)	81
Authentication and session exceptions	
E-3: Detection point tuning analysis considerations (part 3/4)	82
Access control, input, encoding, command injection and file input/output exceptions	
E-4: Detection point tuning analysis considerations (part 4/4)	83
Honey trap, user trend and system trend exceptions, and reputation	

List of figures

Number: Title	Page
0-1: The range of user behavior illustrating that malicious attacks are separate to normal application use	4
1-1: Example application portfolio risk classifications showing the number of applications in each Risk classifications are organization-specific	10
1-2: Example risk classification for five independent applications The applications are completely isolated from each other	10
1-3: Example risk classification for five related applications The applications use a common database and share login credentials (poor practice)	11
1-4: Example risk partitioning of an application If each section is truly isolated from the others, the exposure will affect the resultant risk classification	12
2-1: Detection point categorization overview Using information from the tables in Appendix A	19
2-2: Possible generic pre-processing detection points Based on Figure 2-1 and using information from the Table A-4 in Appendix A; pre-processing detection points highlighted in italicized underlined orange	24
2-3: Example detection point definition overview for an instance of IE2 Initial requirements	28
2-4: Example detection point definition overview for an instance of ACE3 Initial requirements	29
2-5: Example data flow diagram annotated with potential detection points Initial requirements for one process	31
2-6: Normal application use means validation needs to cater for a range of acceptable user data values	32

2-7: Some data unacceptable data values will be much more likely to indicate a malicious user Acceptability is organization and application dependent	33
2-8: Application-layer knowledge increases the ability the ability to differentiate between normal and malicious input	34
2-9: AppSensor Detection Points Inter-Relationships	36
2-10: Example partial detection point schedule for IE2 Partial view of a longer schedule	37
2-11: Example detection point schedule for ACE3	38
3-1: Example schedule of thresholds Partial view of a longer schedule that has many more user event definitions	47
D-1: Request data flow diagram for static image file showing possible detection points User already authenticated	77
D-2: Request data flow diagram for credit card payment showing possible detection points User already authenticated	78
D-3: Request data flow diagram for password change showing possible detection points User already authenticated	79

Appendices

- A Detection point categorization
- B Detection point cross reference with attacks, weaknesses and risks
- C Functional grouping of detection points
- D Example high-level data flow diagrams annotated with detection points
- E Detection point tuning analysis considerations



Appendix A

Detection point categorization

The following four tables categorize the AppSensor detection points by various criteria. See section 2 for a discussion of these.

Table A-1: AppSensor Detection Points Categorized by Signature and Behavior Based Events

Grouped by whether they apply to one user's activity, or potentially all users

Source	Detection Points														Behavior			
	Category	Signature																
One user	Request	RE1	RE2	RE3	RE4	RE5	RE6	RE7	RE8									
	Authentication	AE1	AE2	AE3	AE4	AE5	AE6	AE7	AE8	AE9	AE10	AE11	AE12					
	Session	SE1	SE2	SE3	SE4	SE5	SE6											
	Access Control	ACE1	ACE2	ACE3	ACE4													
	Input Exception	IE1	IE2	IE3	IE4	IE5	IE6											
	Encoding	EE1	EE2															
	Command Injection	CIE1	CIE2	CIE3	CIE4													
	File IO	FIO1	FIO2															
	Honey Trap	HT1	HT2	HT3														
	User Trend														UT1	UT2	UT3	UT4
	Reputation														RP1	RP2	RP3	
	All users	System Trend													STE1	STE2	STE3	
Reputation														RP4				

Source: Detection Points, AppSensor Project v1.1, https://www.owasp.org/images/2/2f/OWASP_AppSensor_Beta_1.1.pdf updated with newer detection points

Table A-2: AppSensor Detection Points Categorised by Suspicious and Attack Events

Grouped by whether the detection points apply to one user's activity, or potentially all users

SOURCE	DETECTION POINTS														
	CATEGORY	SUSPECT			ATTACK										
One user	Request	RE3	RE5	RE6	RE1	RE2	RE4	RE7	RE8						
	Authentication	AE1	AE7		AE2	AE3	AE4	AE5	AE6	AE8	AE9	AE10	AE11	AE12	
	Session	SE3	SE5		SE1	SE2	SE4	SE6							
	Access Control	ACE1	ACE3		ACE2	ACE4									
	Input Exception	IE1	IE2	IE3	IE4	IE5	IE6								
	Encoding	EE1			EE2										
	Command Injection				CIE1	CIE2	CIE3	CIE4							
	File IO	FIO1			FIO2										
	Honey Trap				HT1	HT2	HT3								
	User Trend	UT1	UT2	UT3	UT4										
	Reputation	RP1	RP2	RP3											
All users	System Trend	STE1	STE2	STE3											
	Reputation	RP4													

Source: Appendix B: Response - Suspect vs. Attack Events, AppSensor Project v1.1, https://www.owasp.org/images/2/2f/OWASP_AppSensor_Beta_1.1.pdf updated with newer detection points

Table A-3: AppSensor Detection Points Categorised by Whether They are Discrete, Aggregating or Modifying

Grouped by whether they apply to one user's activity, or potentially all users

Source	Detection Points														
	Category	Discrete									Aggregating			Modifying	
One user	Request	RE1	RE2	RE3	RE4	RE5	RE6	RE7	RE8						
	Authentication	AE4	AE5	AE6	AE7	AE8	AE9	AE10	AE11	AE12	AE1	AE2	AE3		
	Session	SE1	SE2	SE3	SE4						SE5	SE6			
	Access Control	ACE1	ACE2	ACE3	ACE4										
	Input Exception	IE1	IE2	IE3	IE4	IE5	IE6								
	Encoding	EE1	EE2												
	Command Injection	CIE1	CIE2	CIE3	CIE4										
	File IO	FIO1									FIO2				
	Honey Trap	HT1	HT2	HT3											
	User Trend										UT1	UT2	UT3	UT4	
	Reputation													RP1	RP2
All users	System Trend										STE1	STE2	STE3		
	Reputation													RP4	

Table A-4: AppSensor Detection Points Categorised by Generic Pre-Processing vs Business Layer Implementation

Grouped by whether they apply to one user's activity, or potentially all users

Source	Detection Points														
	Category	Generic Pre-Processing								Business Layer					
One user	Request	RE1	RE2	RE3	RE4	RE5	RE6	RE7	RE8						
	Authentication	AE4	AE5	AE6	AE7	AE8	AE9	AE10	AE11	AE1	AE2	AE3	AE12		
	Session	SE1	SE2	SE3	SE4					SE5	SE6				
	Access Control	ACE3								ACE1	ACE2	ACE4			
	Input Exception									IE1	IE2	IE3	IE4	IE5	IE6
	Encoding	EE1	EE2												
	Command Injection	CIE1								CIE2	CIE3	CIE4			
	File IO									FIO1	FIO2				
	Honey Trap	HT2								HT1	HT3				
	User Trend									UT1	UT2	UT3	UT4		
	Reputation									RP1	RP2	RP3			
All users	System Trend									STE1	STE2	STE3			
	Reputation									RP4					

Source: Appendix C: Implementation - Aspect Orientated vs. Business Layer, AppSensor Project v1.1, https://www.owasp.org/images/2/2f/OWASP_AppSensor_Beta_1.1.pdf updated with newer detection points and re-allocated

Appendix B

Detection point cross reference with attacks, weaknesses and risks

The colors shows the applicability of the detection point for the attack/weakness/risk:





	Strong
	Partial
	Slight relevance
	No relevance

Table B-1: WASC Threat Classification attacks cross-reference with detection points (part 1/2)

Request, authentication, session and access control exceptions

ID	Name	Signature Based Events								AuthenticationException												SessionException						AccessControlException			
		RE1	RE2	RE3	RE4	RE5	RE6	RE7	RE8	AE1	AE2	AE3	AE4	AE5	AE6	AE7	AE8	AE9	AE10	AE11	AE12	SE1	SE2	SE3	SE4	SE5	SE6	ACE1	ACE2	ACE3	ACE4
WASC Threat Classification - Attacks																															
WASC-03	Integer Overflows																														
WASC-05	Remote File Inclusion																														
WASC-06	Format String																														
WASC-07	Buffer Overflow																														
WASC-08	Cross-site Scripting																														
WASC-09	Cross-site Request Forgery																														
WASC-10	Denial of Service																														
WASC-11	Brute Force																														
WASC-12	Content Spoofing																														
WASC-18	Credential/Session Prediction																														
WASC-19	SQL Injection																														
WASC-23	XML Injection																														
WASC-24	HTTP Request Splitting																														
WASC-25	HTTP Response Splitting																														
WASC-26	HTTP Request Smuggling																														
WASC-27	HTTP Response Smuggling																														
WASC-28	Null Byte Injection																														
WASC-29	LDAP Injection																														
WASC-30	Mail Command Injection																														
WASC-31	OS Commanding																														
WASC-32	Routing Detour																														
WASC-33	Path Traversal																														
WASC-34	Predictable Resource Location																														
WASC-35	SOAP Array Abuse																														
WASC-36	SSI Injection																														
WASC-37	Session Fixation																														
WASC-38	URL Redirector Abuse																														
WASC-39	XPath Injection																														
WASC-41	XML Attribute Blowup																														
WASC-42	Abuse of Functionality																														
WASC-43	XML External Entities																														
WASC-44	XML Entity Expansion																														
WASC-45	Fingerprinting																														
WASC-46	XQuery Injection																														

Table B-2: WASC Threat Classification attacks cross-reference with detection points (part 2/2)

Input, encoding, command injection, file input/output, honey trap, user trend and system trend exceptions, and reputation

ID	Name	InputException															Behavior Based Events														
		IE1	IE2	IE3	IE4	IE5	IE6	EE1	EE2	CIE1	CIE2	CIE3	CIE4	FIO1	FIO2	HT1	HT2	HT3	UT1	UT2	UT3	UT4	STE1	STE2	STE3	RP1	RP2	RP3	RP4		
WASC Threat Classification - Attacks																															
WASC-03	Integer Overflows																														
WASC-05	Remote File Inclusion																														
WASC-06	Format String																														
WASC-07	Buffer Overflow																														
WASC-08	Cross-site Scripting																														
WASC-09	Cross-site Request Forgery																														
WASC-10	Denial of Service																														
WASC-11	Brute Force																														
WASC-12	Content Spoofing																														
WASC-18	Credential/Session Prediction																														
WASC-19	SQL Injection																														
WASC-23	XML Injection																														
WASC-24	HTTP Request Splitting																														
WASC-25	HTTP Response Splitting																														
WASC-26	HTTP Request Smuggling																														
WASC-27	HTTP Response Smuggling																														
WASC-28	Null Byte Injection																														
WASC-29	LDAP Injection																														
WASC-30	Mail Command Injection																														
WASC-31	OS Commanding																														
WASC-32	Routing Detour																														
WASC-33	Path Traversal																														
WASC-34	Predictable Resource Location																														
WASC-35	SOAP Array Abuse																														
WASC-36	SSI Injection																														
WASC-37	Session Fixation																														
WASC-38	URL Redirector Abuse																														
WASC-39	XPath Injection																														
WASC-41	XML Attribute Blowup																														
WASC-42	Abuse of Functionality																														
WASC-43	XML External Entities																														
WASC-44	XML Entity Expansion																														
WASC-45	Fingerprinting																														
WASC-46	XQuery Injection																														

Table B-3: WASC Threat Classification weaknesses and OWASP Top Ten Risks cross-reference with detection points (part 1/2)

Request, authentication, session and access control exceptions

ID	Name	Signature Based Events								AuthenticationException												SessionException						AccessControlException			
		RE1	RE2	RE3	RE4	RE5	RE6	RE7	RE8	AE1	AE2	AE3	AE4	AE5	AE6	AE7	AE8	AE9	AE10	AE11	AE12	SE1	SE2	SE3	SE4	SE5	SE6	ACE1	ACE2	ACE3	ACE4
WASC Threat Classification - Attacks																															
WASC-03	Integer Overflows																														
WASC-05	Remote File Inclusion																														
WASC-06	Format String																														
WASC-07	Buffer Overflow																														
WASC-08	Cross-site Scripting																														
WASC-09	Cross-site Request Forgery																														
WASC-10	Denial of Service																														
WASC-11	Brute Force																														
WASC-12	Content Spoofing																														
WASC-18	Credential/Session Prediction																														
WASC-19	SQL Injection																														
WASC-23	XML Injection																														
WASC-24	HTTP Request Splitting																														
WASC-25	HTTP Response Splitting																														
WASC-26	HTTP Request Smuggling																														
WASC-27	HTTP Response Smuggling																														
WASC-28	Null Byte Injection																														
WASC-29	LDAP Injection																														
WASC-30	Mail Command Injection																														
WASC-31	OS Commanding																														
WASC-32	Routing Detour																														
WASC-33	Path Traversal																														
WASC-34	Predictable Resource Location																														
WASC-35	SOAP Array Abuse																														
WASC-36	SSI Injection																														
WASC-37	Session Fixation																														
WASC-38	URL Redirector Abuse																														
WASC-39	XPath Injection																														
WASC-41	XML Attribute Blowup																														
WASC-42	Abuse of Functionality																														
WASC-43	XML External Entities																														
WASC-44	XML Entity Expansion																														
WASC-45	Fingerprinting																														
WASC-46	XQuery Injection																														

Table B-4: WASC Threat Classification weaknesses and OWASP Top Ten Risks cross-reference with detection points (part 2/2)

Input, encoding, command injection, file input/output, honey trap, user trend and system trend exceptions, and reputation

ID	Name	InputException															Behavior Based Events														
		IE1	IE2	IE3	IE4	IE5	IE6	EE1	EE2	CIE1	CIE2	CIE3	CIE4	FIO1	FIO2	HT1	HT2	HT3	UT1	UT2	UT3	UT4	STE1	STE2	STE3	RP1	RP2	RP3	RP4		
WASC Threat Classification - Attacks																															
WASC-03	Integer Overflows																														
WASC-05	Remote File Inclusion																														
WASC-06	Format String																														
WASC-07	Buffer Overflow																														
WASC-08	Cross-site Scripting																														
WASC-09	Cross-site Request Forgery																														
WASC-10	Denial of Service																														
WASC-11	Brute Force																														
WASC-12	Content Spoofing																														
WASC-18	Credential/Session Prediction																														
WASC-19	SQL Injection																														
WASC-23	XML Injection																														
WASC-24	HTTP Request Splitting																														
WASC-25	HTTP Response Splitting																														
WASC-26	HTTP Request Smuggling																														
WASC-27	HTTP Response Smuggling																														
WASC-28	Null Byte Injection																														
WASC-29	LDAP Injection																														
WASC-30	Mail Command Injection																														
WASC-31	OS Commanding																														
WASC-32	Routing Detour																														
WASC-33	Path Traversal																														
WASC-34	Predictable Resource Location																														
WASC-35	SOAP Array Abuse																														
WASC-36	SSI Injection																														
WASC-37	Session Fixation																														
WASC-38	URL Redirector Abuse																														
WASC-39	XPath Injection																														
WASC-41	XML Attribute Blowup																														
WASC-42	Abuse of Functionality																														
WASC-43	XML External Entities																														
WASC-44	XML Entity Expansion																														
WASC-45	Fingerprinting																														
WASC-46	XQuery Injection																														

Appendix C

Functional grouping of detection points

In the following seven tables, colours are used to indicate which discrete detection points for the main types:

Detailed checks depending on the context, functionality, inputs and outputs:

- detailed authorisation
- detailed request validation
- detailed result inspection

Authentication

- log in
- Password reset by an unauthenticated user
- Password change by an authenticated user

The key to the colors is:




	Most appropriate detection points
	Other detection point may also be suitable for additional aspects, or provide partial coverage
	No relevance

Table C-1: Broad request checks

Relevant to all entry points

SOURCE	DETECTION POINTS												AGGREGATING				MODIFYING		
	CATEGORY	DISCRETE																	
One user	Request	RE1	RE2	RE3	RE4	RE5	RE6	RE7	RE8										
	Authentication	AE4	AE5	AE6	AE7	AE8	AE9	AE10	AE11	AE12	AE1	AE2	AE3						
	Session	SE1	SE2	SE3	SE4						SE5	SE6							
	Access Control	ACE1	ACE2	ACE3	ACE4														
	Input Exception	IE1	IE2	IE3	IE4	IE5	IE6												
	Encoding	EE1	EE2																
	Command Injection	CIE1	CIE2	CIE3	CIE4														
	File IO	FIO1									FIO2								
	Honey Trap	HT1	HT2	HT3															
	User Trend										UT1	UT2	UT3	UT4					
	Reputation																RP1	RP2	RP3
All users	System Trend										STE1	STE2	STE3						
	Reputation																RP4		

Examples:

- checking the request meets expected HTTP standard and does not contain unexpected encoding
- checking the requested URL is an allowable entry point to the application
- ensuring the session ID is valid and current for the user
- ensuring all the required parameters for the entry point are provided and their type (e.g. integer, short string), and nothing extra is provided or duplicated (but not the detailed validation of those values)
- checking an XML payload against a schema.

Table C-2: Detailed authorisation (e.g. validation beyond general permissions to script)

Possible detection points in addition to those in Table C-1 for broad request checks, relevant to all entry points

Source	Detection Points										Aggregating				Modifying		
	Category	Discrete															
One user	Request	RE1	RE2	RE3	RE4	RE5	RE6	RE7	RE8								
	Authentication	AE4	AE5	AE6	AE7	AE8	AE9	AE10	AE11	AE12	AE1	AE2	AE3				
	Session	SE1	SE2	SE3	SE4						SE5	SE6					
	Access Control	ACE1	ACE2	ACE3	ACE4												
	Input Exception	IE1	IE2	IE3	IE4	IE5	IE6										
	Encoding	EE1	EE2														
	Command Injection	CIE1	CIE2	CIE3	CIE4												
	File IO	FIO1									FIO2						
	Honey Trap	HT1	HT2	HT3													
	User Trend										UT1	UT2	UT3	UT4			
	Reputation															RP1	RP2
All users	System Trend										STE1	STE2	STE3				
	Reputation															RP4	

Examples:

- checking the user has permissions to requested resource
- checking the action is allowed by the user
- checking parameter values to ensure they have permission to the target object.

Table C-3: Detailed request validation

Possible detection points in addition to those in Tables C-1 and C-2 for broad request checks and detailed authorisation checks

Source	Detection Points										Aggregating							Modifying		
	Category	Discrete																		
One user	Request	RE1	RE2	RE3	RE4	RE5	RE6	RE7	RE8											
	Authentication	AE4	AE5	AE6	AE7	AE8	AE9	AE10	AE11	AE12	AE1	AE2	AE3							
	Session	SE1	SE2	SE3	SE4						SE5	SE6								
	Access Control	ACE1	ACE2	ACE3	ACE4															
	Input Exception	IE1	IE2	IE3	IE4	IE5	IE6													
	Encoding	EE1	EE2																	
	Command Injection	CIE1	CIE2	CIE3	CIE4															
	File IO	FIO1									FIO2									
	Honey Trap	HT1	HT2	HT3																
	User Trend										UT1	UT2	UT3	UT4						
	Reputation															RP1	RP2	RP3		
All users	System Trend										STE1	STE2	STE3							
	Reputation															RP4				

Examples:

- consistency of parameters provided based on the business logic (e.g. some parameters are optional, but must be provided when another parameter has a particular value)
- parameter values contravene input validation checks
- checking the values of nodes within an XML file.

Table C-4: Detailed result inspection

Possible detection points in addition to those in Tables C-1, C-2 and C-3

SOURCE	DETECTION POINTS										AGGREGATING				MODIFYING		
	CATEGORY	DISCRETE															
One user	Request	RE1	RE2	RE3	RE4	RE5	RE6	RE7	RE8								
	Authentication	AE4	AE5	AE6	AE7	AE8	AE9	AE10	AE11	AE12	AE1	AE2	AE3				
	Session	SE1	SE2	SE3	SE4						SE5	SE6					
	Access Control	ACE1	ACE2	ACE3	ACE4												
	Input Exception	IE1	IE2	IE3	IE4	IE5	IE6										
	Encoding	EE1	EE2														
	Command Injection	CIE1	CIE2	CIE3	CIE4												
	File IO	FIO1									FIO2						
	Honey Trap	HT1	HT2	HT3													
	User Trend										UT1	UT2	UT3	UT4			
	Reputation														RP1	RP2	RP3
All users	System Trend										STE1	STE2	STE3				
	Reputation														RP4		

Examples:

- use of a parameter leads to an unexpected condition such as more than one record in a database result set, when only one is expected
- calculated hash values of records in the security event log are no longer correct
- a database table's foreign key has been deleted.

Table C-5: Log in

Possible authentication-specific detection points in addition to those in Tables C-1, C-2 and C-3

Source	Detection Points																			
	Category	Discrete								Aggregating				Modifying						
One user	Request	RE1	RE2	RE3	RE4	RE5	RE6	RE7	RE8											
	Authentication	AE4	AE5	AE6	AE7	AE8	AE9	AE10	AE11	AE12	AE1	AE2	AE3							
	Session	SE1	SE2	SE3	SE4						SE5	SE6								
	Access Control	ACE1	ACE2	ACE3	ACE4															
	Input Exception	IE1	IE2	IE3	IE4	IE5	IE6													
	Encoding	EE1	EE2																	
	Command Injection	CIE1	CIE2	CIE3	CIE4															
	File IO	FIO1								FIO2										
	Honey Trap	HT1	HT2	HT3																
	User Trend										UT1	UT2	UT3	UT4						
	Reputation														RP1	RP2	RP3			
All users	System Trend										STE1	STE2	STE3							
	Reputation														RP4					

Examples:

- username and password client-side validation has been bypassed
- required parameters missing
- additional POST parameters received.

Table C-6: Password reset by an unauthenticated user

Possible authentication-specific detection points in addition to those in Tables C-1, C-2 and C-3

Source	Detection Points										Aggregating				Modifying		
	Category	Discrete															
One user	Request	RE1	RE2	RE3	RE4	RE5	RE6	RE7	RE8								
	Authentication	AE4	AE5	AE6	AE7	AE8	AE9	AE10	AE11	AE12	AE1	AE2	AE3				
	Session	SE1	SE2	SE3	SE4						SE5	SE6					
	Access Control	ACE1	ACE2	ACE3	ACE4												
	Input Exception	IE1	IE2	IE3	IE4	IE5	IE6										
	Encoding	EE1	EE2														
	Command Injection	CIE1	CIE2	CIE3	CIE4												
	File IO	FIO1									FIO2						
	Honey Trap	HT1	HT2	HT3													
	User Trend										UT1	UT2	UT3	UT4			
	Reputation														RP1	RP2	RP3
All users	System Trend										STE1	STE2	STE3				
	Reputation														RP4		

Examples:

- username field submitted empty
- invalid characters included in the username provided
- username does not match the expected pattern.

Table C-7: Password change by an authenticated user

Possible authentication-specific detection points in addition to those in Tables C-1, C-2 and C-3

Source	Detection Points															
	Category	Discrete								Aggregating				Modifying		
One user	Request	RE1	RE2	RE3	RE4	RE5	RE6	RE7	RE8							
	Authentication	AE4	AE5	AE6	AE7	AE8	AE9	AE10	AE11	AE12	AE1	AE2	AE3			
	Session	SE1	SE2	SE3	SE4						SE5	SE6				
	Access Control	ACE1	ACE2	ACE3	ACE4											
	Input Exception	IE1	IE2	IE3	IE4	IE5	IE6									
	Encoding	EE1	EE2													
	Command Injection	CIE1	CIE2	CIE3	CIE4											
	File IO	FIO1									FIO2					
	Honey Trap	HT1	HT2	HT3												
	User Trend										UT1	UT2	UT3	UT4		
	Reputation														RP1	RP2
All users	System Trend										STE1	STE2	STE3			
	Reputation														RP4	

Examples:

- password value longer than the maxlength attribute for the form element
- hidden form field missing.

Appendix D

Example high-level data flow diagrams annotated with detection points

The following three data flow diagrams showing possible detection points.

Figurer D-2 is a duplication of Figure 2-5 in the main text.

Figure D-1: Request data flow diagram for static image file showing possible detection points

User already authenticated

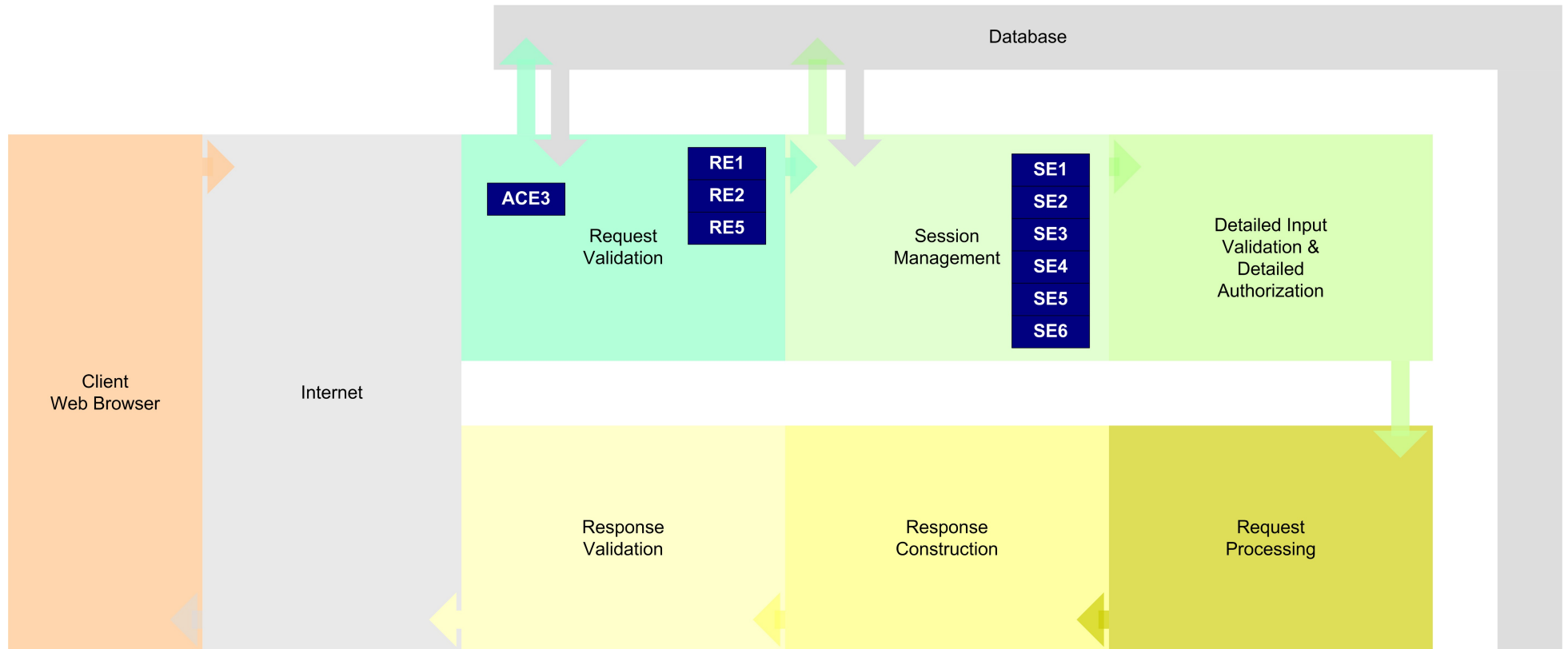


Figure D-2: Request data flow diagram for credit card payment showing possible detection points

User already authenticated

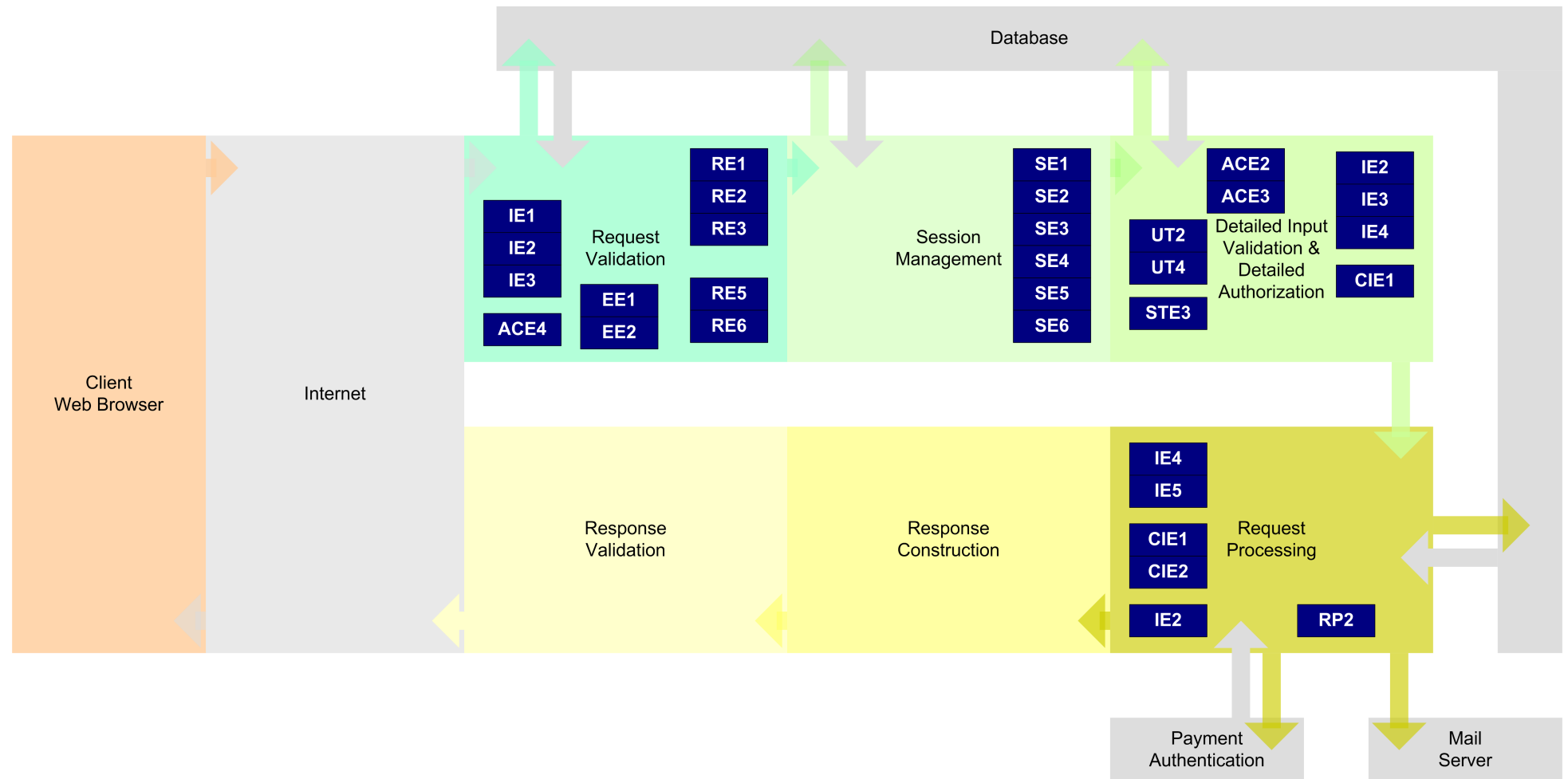
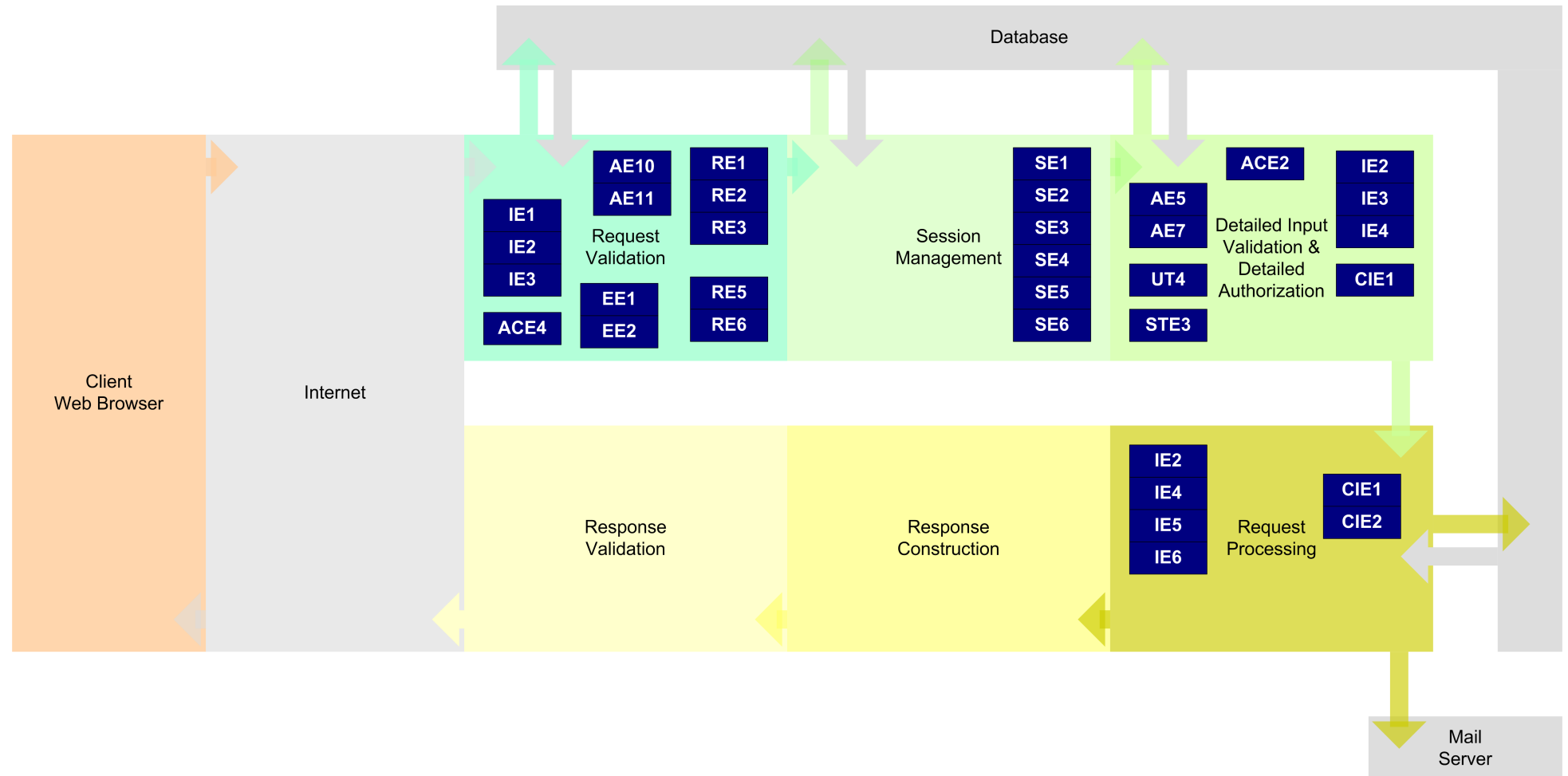


Figure D-3: Request data flow diagram for password change showing possible detection points

User already authenticated



Appendix E

Detection point tuning analysis considerations

Some detection points may be susceptible to an increased number of false positives if implemented too strictly. Some suggestions for analysis during *step 2d* are below.

Table E-1: Detection point tuning analysis considerations (part 1/4)

Request exceptions

DETECTION POINT CATEGORY	ID	CONSIDERATIONS DESCRIPTION	REFERENCES
Request Exception	RE1	Browsers and proxies using the HEAD method to check whether the content of a file has changed	HTTP/1.1: Method Definitions http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
	RE2	-	-
	RE3	Some resources may allow both GET and POST methods e.g. and edit form may be hyperlinked using a parameter value defining the record to be edited, but the form is submitted by POST to itself. Users may bookmark a page that is the result of a POST and return to it at a later date.	-
	RE4	As RE3 above	-
	RE5	Links from third party sites/services may included additional parameters (e.g. from search engines, from advertisements). Additional cookies headers may be added by other applications or by third parties such as advertisers, and there may be very little control over these. Additional HTTP headers may be added by intermediate network devices (e.g. for traffic management).	-
	RE6	A bookmarked page may be missing the required POST parameters (see also RE3). Users may choose to send a blank or different User Agent header value.	-
	RE7	If the input field does not have client-side validation and/or MAXLENGTH attributes, a user might inadvertently copy in some text that is longer than expected.	-
	RE8	Text fields may include text from copy and paste operations that contain illegal characters.	-

Table E-2: Detection point tuning analysis considerations (part 2/4)

Authentication and session exceptions

DETECTION POINT CATEGORY	ID	CONSIDERATIONS DESCRIPTION	REFERENCES
Authentication Exception	AE1	-	-
	AE2	A users providing the same wrong password more than once may be different to different wrong passwords.	Account Lockout, Bill Cheswick, Episode 76, OWASP Podcast, September 22, 2010 http://www.owasp.org/index.php/OWASP_Podcast#tab=Latest_Shows
	AE3	-	-
	AE4	As RE7 above	-
	AE5	As RE7 above.	-
	AE6	Users may be confused between a username, customer identification code and their account number, or even between offline and online identifiers. Mis-typing might add a character like "]" or "#" if these are adjacent to the ENTER/CR key. Whitespace may be appended to values when copied from a spreadsheet cell (e.g. a line feed character when cell values are copied and pasted from Excel). A password may be put in the username field by accident.	-
	AE7	As AE6 above.	-
	AE8	-	-
	AE9	-	-
	AE10	See RE5.	-
	AE11	See RE6.	-
	AE12	Common usernames might be allowed during self-registration or when editing account details.	-
Session Exception	SE1	In a poorly designed application, the length of the cookie value, or the combined size of all the cookies, might possibly exceed that which is supported.	-
	SE2	-	-
	SE3	In a poorly designed application, the number of cookies might exceed the allowed number supported by the user's browser.	-
	SE4	A mis-configured proxy might send the same session ID or cookie for all users.	-
	SE5	If the full IP address is used for this, it may change slightly from request to request by the same user.	-
	SE6	-	-

Table E-3: Detection point tuning analysis considerations (part 3/4)

Access control, input, encoding, command injection and file input/output exceptions

DETECTION POINT	CONSIDERATIONS		
CATEGORY	ID	DESCRIPTION	REFERENCES
Access Control Exception	ACE1	Bookmarking , truncation, and mistyping issues could lead to some access control exceptions.	-
	ACE2	As ACE1 above for bookmarking.	-
	ACE3	Requests for non-existent resources may occur for many reasons such as	Benign Unexpected URLs - Part 1 - Missing (404 Not Found Error) Files http://www.clerkendweller.com/2010/10/26/Benign-Unexpected-URLs-Part-1-Missing-Files
	ACE4	-	-
Input Exception	IE1	There are many patterns which could be XSS but may also be normal user input to a free text field e.g. "Press the 'drop' button" if a pattern were looking for a single quotation mark followed by SQL commands like DROP, INSERT, UPDATE and DELETE. Applications that are used to discuss or share information about programming, software development and security may want to allow such free text input, provided it is encoded/escaped correctly.	-
	IE2	As IE1 above.	-
	IE3	As IE1 above.	-
	IE4	This detection point should only be used with parameters that cannot be altered by accident. Input types text and textarea would not normally be suitable, even if the elements are disabled in the browser. Be wary of assuming JavaScript will prevent modification of form elements in all conditions.	-
	IE5	-	-
	IE6	-	-
Encoding Exception	EE1	Data supplied by other party systems may have encoding issues.	-
	EE2	As EE2 above.	-
Command Injection Exception	CIE1	As IE1 above.	-
	CIE2	-	-
	CIE3	-	-
	CIE4	-	-
File IO Exception	FIO1	-	-
	FIO2	-	-

Table E-4: Detection point tuning analysis considerations (part 4/4)

Honey trap, user trend and system trend exceptions, and reputation

DETECTION POINT		CONSIDERATIONS	
CATEGORY	ID	DESCRIPTION	REFERENCES
Honey Trap	HT1	-	-
	HT2	-	-
	HT3	-	-
User Trend Exception	UT1	Use of bookmarked URLs and the "back" button might generate out-of-sequence requests. See also related frequency of feature use in UT4 below.	-
	UT2	Time periods need to be set broadly enough to cater for the normal spread in user behavior. Some users may use automated tools that store passwords securely to populate and submit authentication forms.	-
	UT3	Some users may correctly change their behavior in the frequency of accessing the site.	-
	UT4	It may be valid for some functionality may be requested repeatedly. For example a real customer placing many orders, a press officer publishing a backlog of press releases, or an administrator populating a staff directory.	-
System Trend Exception	STE1	-	-
	STE2	-	-
	STE3	A special requirement, situation or event may dramatically change the rate of use of certain transactions. See also UT4 above.	-
Reputation	RP1	The currency and accuracy of needs to be considered when the information is used in AppSensor. The method of challenge and removal of inaccuracies, and the speed of this process, should also be considered.	-
	RP2	The level of trust in information from the external device/system/organization needs to be considered.	-
	RP3	-	-
	RP4	The detection point could receive specially crafted input from an attacker, and therefore the information should be considered as untrusted.	-