

Compliance factors

There is a strong reality today in the realm of compliance. You will find yourself interfacing with either auditors or compliance auditing software. The interfacing with auditors is straightforward, they will ask you questions and try to identify where your findings for a given web application target overlap with compliance criteria to form a gap or deficit that needs to be addressed. From the software perspective there is unfortunately no standard and so you will obviously have to be flexible for now. The Application Vulnerability Description Language (AVDL - <http://www.avdl.org>) may be a step towards a solution. Basically you will want every block in your Risk Matrix to be portable to AVDL or some such standard. For the purposes of this exposure we will stick with AVDL. It would then be the job of the target compliance software system to do the correlation between your findings and the respective compliance criteria to identify the compliance related deficiencies for the target organization. An example of some AVDL XML data (based on the earlier example block "Miscrosoft ASP.NET Debugging Enabled") is:

```
<avdl version="1.0"
xmlns="urn:oasis:names:tc:avdl:0.0:mailto:avdl@oasisopen.
org?:avdl:2003-09-27:a" xmlns:xhtml="http://www.w3.org/1999/xhtml"
xmlns:avdln="urn:oasis:names:tc:avdl:0.0:names:mailto:avdl@oasisopen.
org?:2003-09-27" xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
<vulnerability-probe id="5286" time-stamp="2006-01-31T23:46:58">
<test-probe><http-probe>
<request method="DEBUG" connection="" host="<target>:80" requesturi="/
path/startup.aspx" version="HTTP/1.0">
<raw>DEBUG /path/startup.aspx HTTP/1.0
Referer: http://<ref_target>:80/path/
Connection: Close
Host: <target>
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Pragma: no-cache
Content-Length: 0
Command: stop-debug
Connection: closed
Cookie: ASPSESSIONIDAABQTDQT=CCEBGKPCDIBMFILHDHCHJBF;
ASP.NET_SessionId=5midlh55bqdr00fcd512dp45
</raw><parsed><header name="Cookie" value="
ASPSESSIONIDAABQTDQT=CCEBGKPCDIBMFILHDHCHJBF;
ASP.NET_SessionId=5midlh55bqdr00fcd512dp45"/>
<header name="Referer" value="http://<ref_target>:80/path/" />
<header name="Connection" value="Close" />
<header name="Host" value="<target>" />
<header name="User-Agent" value="Mozilla/4.0 (compatible; MSIE 5.01;
Windows NT 5.0)" />
<header name="Pragma" value="no-cache" />
<header name="Content-Length" value="0" />
<header name="Command" value="stop-debug" />
<header name="Connection" value="closed" />
<query value="" /><content value="" />
</parsed></request>
<response>
<raw>
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Sat, 09 Jul 2005 00:12:51 GMT
X-Powered-By: ASP.NET
X-AspNet-Version: 1.1.4322
Cache-Control: private
Content-Type: text/html; charset=utf-8
```

Content-Length: 2

OK

</raw>

<parsed><statusline value="HTTP/1.1 200 OK"/>

<header name="Server" value="Microsoft-IIS/5.0"/>

<header name="Date" value="Sat, 09 Jul 2005 00:12:51 GMT"/>

<header name="X-Powered-By" value="ASP.NET"/>

<header name="X-AspNet-Version" value="1.1.4322"/>

<header name="Cache-Control" value="private"/>

<header name="Content-Type" value="text/html; charset=utf-8"/>

<header name="Content-Length" value="2"/>

<content/></parsed></response></http-probe></test-probe>

<vulnerability-description title="Microsoft ASP.NET Debugging Enabled">

<summary>Microsoft ASP.NET Debugging Enabled</summary>

<description>A custom HTTP verb exists which allows a remote user to enable debugging support in ASP.NET. This issue affects every folder, since each folder is treated individually as a separate project and has its own global.asax file. If the verb is 'DEBUG', the debug handler is loaded in place of the URL that was requested. If debugging is enabled, ASP.NET looks for a header called 'Command', whose value can be of two types

 Command: stop-debug

 Command: start-debug

Depending upon the access control present on the server, a remote debug session can potentially disclose information about the target system as well as information about the target web application.</description>

<classification xmlns:was="urn:oasis:names:tc:was:1.0:..."

name="was:severity" value="25"/>

<recommendation><user-description><description/></userdescription></

recommendation>

<test-script id="test-script-2">

<declare name="path" type="string" default="/path/startup.aspx"/>

<declare name="protocol" type="string" default="HTTP/1.1"/>

<declare name="host" type="host" default="<target>"/>

<declare name="port" type="integer" default="80"/>

<sequence><http-transaction>

<request xmlns="urn:oasis:names:tc:avdl:0.0:mailto:avdl@oasisopen.

org?:avdl:2003-09-27:a">DEBUG

<var name="path"/>

<var name="protocol"/>

Referer: http://<var name="host"/>:<var name="port"/>/path/

Connection: Close

Host: <var name="host"/>

User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)

Pragma: no-cache

Content-Length: 0

Command: stop-debug

Connection: closed

Cookie: ASPSESSIONIDAABQTDQT=CCEBGKPCDMIBMFILHDHCHJBF;

ASP.NET_SessionId=5midlh55bqdr00fcd512dp45

</request>

<response>

<expect status-code="200" reason-phrase="OK"/>

</response>

</http-transaction></sequence></test-script></vulnerabilitydescription></

vulnerability-probe>

...

</avdl>

That data you generate can be fed into software systems that will know how to parse it and correlate it to the compliance criteria at hand. Other times you may need to pump that data in to some system manually. There are many of these software packages spawning off out there. One example of such a target system is a web-based solution (entitled “The Guard”) by Roland Cozzolino’s Compliancy Group, LLC. (<http://www.compliancygroup.com>). The Guard knows how to parse through Nessus results as well as other sources of data (such as AVDL) so that it can automatically tell the target where it will fail a compliance audit in respects to the technical points related to your findings. In order for you to appreciate the complexity at hand take a look at a brief example.

This example will feature a simple and common attack via a dictionary hack. The goal is to show how the results from a pen test can be correlated directly to regulations. Health Insurance Portability and Accountability Act (HIPAA) will be used for this example. One key point to be aware of is that any discovered vulnerability can cause breakdowns across multiple regulations. But for exemplary purposes this will be kept as simple as possible. To fully understand the relationship between Pen Testing and Compliance Management, a brief description of HIPAA and some of its standards is required. HIPAA’s technical goal is to protect sensitive patient information from unauthorized access. This example will violate the following HIPAA rules:

Regulation Basic Interpretation of Standard

§ 164.308.a.3 Workforce Security	Ensure that people have appropriate access to protected data and be able to terminate access to this information when appropriate.
§ 164.308.a.6 Security Incident Procedures	Identify and respond to suspected or known security incidents and document their outcomes.
§ 164.312.a Access Control	Assign unique identifiers to every user that accesses systems with patient information to ensure their access can be tracked and validated. Furthermore, store and transmit patient information in an encrypted format.
§ 164.312.b Audit Controls	Implement hardware, software, and/or procedural mechanisms that record and examine activity in information systems that contain or use electronic protected health information.

Please note that the descriptions in the table above are simplistic interpretations of the regulations and are only intended to exemplify the parallels between Pen Testing and Compliance Management.

In order to give you a feel for a real environment this example will be constructed against a mock company, the AB321 Trust Fund (AB321TF). Envision an organization with 50,000 members. AB321TF is self-insured, thus its participants get medical insurance directly from it. Due to the nature of its business, and the self-insured status, both medical and financial records are stored in a DB inside AB321TF’s LAN. This DB is front-ended with a website that allows for viewing and modification of the data stored in it. This includes sensitive personal data (phone number, address, etc.). The web server is exposed directly to the Internet and the staff at AB321TF feels safe because the app requires user authentication for access and runs over HTTPS. Clearly this sense of security has no correlation to the legal regulations that AB321TF must comply with due to the nature of the data they store.

Jay Smith is this examples fictitious pen tester hired by AB321TF to work his magic. The results of his work will establish a security posture as well as an important element in the compliance picture for AB321TF. Jay starts off by launching a simple dictionary attack on the authentication with Brutus. The tools diligently cycles through thousands of username/password combinations. And eventually there is a hit on an account used for trivial administrative backup purposes. Because multiple people handle the backup processes on the application and its data the password is easy to remember so that people can do their work. In order for the personnel to remember this password it is kept weak and broken by the data Jay fed through Brutus.

Let's say the account he gets in with has access to virtually nothing in terms of application functionality. Although the account limits Jay, it does give him access to the web application postauthentication. So he can see some HTML forms that unauthenticated users cannot see. Instinctually he starts poking at these forms for SQL Injection vulnerabilities. Through this technique he takes advantage of the lack of input validation in place and he gathers data about the DB table structures and he actually gets to see more data than he should be seeing. Let's say he uses the applications established connection to the DB to pull down all the information in the *user* table (which contains username, email, password, SSN, address, phone, etc.).

The entire attack just took place via the browser and there is no use of sophisticated tools, just a knowledgeable Jay. So he spends sometime executing similar attacks on a variety of other servers on AB321TF's LAN and documents all of his findings. He must in turn correlate these findings so that they establish the compliance posture that AB321TF management needs to be aware of.

This trivial example exemplifies how quickly and easily an entity can be in violation of regulatory standards. A brief example of Jay's findings correlated to compliance regulations is listed in the table below.

Regulation Description of Violation

§ 164.308.a.3	The account that was used to break in should have been removed from the web server upon completion of the job.
§ 164.308.a.6	Nobody knew the system was compromised and, as a result, the threat was not addressed in a timely fashion.
§ 164.312.a	When a user accesses their own information, a log is kept and their respective IDs are retained in the system via application logs. However, when directly connecting to the database from the web server, no controls are built in to validate who the person is requesting the information. In fact, the database logs only show that the web server made a request. Furthermore, the data is not stored in an encrypted format on the server, and can thus be read as plain text via simple SQL select statements.
§ 164.312.b	The database allowed the access based on the fact that the web server requested the information and never checked the IP address of the client machine accessing it.

Figure 9-3 of The Guard exemplifies how these types of pen-testing results get correlated with regulations in real world environment.

Figure 9-3

The screenshot shows the 'The Guard' compliance application interface. At the top, there is a navigation bar with tabs: Home, Define, Discovery, Audit, Remediate, Track, and Report. The main content area is titled 'Define Regulatory Gap' and includes fields for Auditor (Jay Smith), Gap Title (Security hole on web server exposes backend machines), Gap Type (Process Gap), Department Designation (MIS), and Assessment Date (1/15/2006). Below these fields are tabs for Regulations, Gap Details, Supporting Docs, and Related Items. The 'Regulations' tab is active, showing a list of regulatory standards with checkboxes: 164.308.a.3 : Workforce Security, 164.308.a.6 : Security Incident Procedures, 164.312.a : Access Control, 164.312.b : Audit Controls, and 164.312.d : Person or Entity Authentication. To the right of this list is a text area containing a detailed description of a security hole found during a dictionary hack on a corporate web server, including steps taken to reproduce the behavior. Below the regulations list are buttons for 'Save Information' and 'Clear All Fields'. The bottom section is titled 'Modify/View Gap Items' and shows a list of gap items with dates and status: '12/12/2005 : No policies defining roles and responsibilities of Security Officer (Remediated)' and '12/18/2005 : Workstations in datacenter don't automatically log users off and remain exposed to tampering (Open)'. Buttons for 'View Selected Item' and 'Delete Selected Item' are at the bottom. The footer contains copyright information for Compliance Group, LLC.

Compliance Group
"Become Compliant. Stay Compliant."
The Guard
guarding your organization from regulatory infractions

Home Define Discovery Audit Remediate Track Report

Define Regulatory Gap General Hospital

Auditor: Jay Smith
Gap Title: Security hole on web server exposes backend machines
Gap Type: Process Gap
Department Designation: MIS
Assessment Date: 1/15/2006

Regulations Gap Details Supporting Docs Related Items

View selected regulations only (filtered)

- 164.308.a.3 : Workforce Security
- 164.308.a.6 : Security Incident Procedures
- 164.312.a : Access Control
- 164.312.b : Audit Controls
- 164.312.d : Person or Entity Authentication

We ran a basic dictionary hack on corporate web server. Upon successfully accessing the server, POST information was scanned, eventually providing enough information to perform basic SQL injection. Please see attached documents under supporting docs for screenshots and custom script which echoed out user information. Attached documents include:

1. userlisting.sql
2. dictionary.pl
3. userdata-screenshot.jpg

Resulting information and gap specifics are located in

Save Information Clear All Fields

Modify/View Gap Items

** View All Gap Types ** View Remediated and Non-Remediated Gaps

- 12/12/2005 : No policies defining roles and responsibilities of Security Officer (Remediated)
- 12/18/2005 : Workstations in datacenter don't automatically log users off and remain exposed to tampering (Open)

View Selected Item Delete Selected Item

The Guard - copyright © 2005 Compliance Group, LLC. All Rights Reserved
DHTML Menu / JavaScript Menu Powered By OpenCube

Association of the findings with both the auditor and the specific department(s) in question to streamline remediation.

Association of the findings to one or more regulatory standards.

Ability to load up previously saved findings for reference or modification.

Quick and easy access to regulations and their full descriptions.

Ability to upload scripts, screen shots or any other additional documentation to support your findings, as well as provide in-depth descriptions of the problem as it pertains to the regulations.

Application should have a simple way to input how each test was performed, what information was collected and the steps taken to replicate the behavior.

The bottom line is that the data you generate will serve multiple purposes, compliance with regulations is just one of them. In Chapter 10 you will see edge level remediation tactics that can also benefit from this same data where rulesets can be generated from your discovered vulnerabilities.