Analysis Results

WebGoat-main.zip

Report Date
2024-09-10 20:35:51
Report Author
admin
Classification Method
HIPAA
Product Version
11.0-SNAPSHOT+f428841a
Rules Version
11-SNAPSHOT.130321

Confidentiality Note

This report is intended only for the person(s) or entity to which it is addressed and contains confidential and privileged information. If you are not the intended recipient, you must not view, use, copy, disclose, or otherwise disseminate this report or any part of it. Doing so is strictly prohibited, and may result in legal proceedings. If you received this in error, please notify the sender immediately and destroy any copies of this information.

01 Project Information

Project Name

WebGoat-main.zip

UUID

185e14e9-0a35-4d27-8a31-4f6275f3bdcc

Project in DerScanner

Dynamics by vulnerabilities

Vulnerabilities are divided by severity level: critical, medium, low and info.

CRITICAL LEVEL	MEDIUM LEVEL	LOW LEVEL	INFORMATION
Likely to lead to compromise confidential data and violation of the integrity of the system.	May be less likely to lead to compromising confidential data and violating the integrity of the system, or are less serious security	Can become a potential security risk.	Signal a violation of good programming practice.

First of all, pay attention to vulnerabilities of critical and medium levels.

Dynamics by rating

The app score is calculated on a scale from 0 to 5. Score is calculated based on the number of critical and medium level vulnerabilities. The impact of critical vulnerabilities is greater than that of medium level vulnerabilities, and does not take into account the amount of code. Medium level vulnerabilities are taken into account based on their frequency and total number of source code lines.

Scan History

	Date and Time	Status	Languages	Lines of Code	Number of Vulnerabilities					Score
					Critical	Medium	Low	Info	Total	
1/1	2024-09-04 09:36:07	completed	Config files, JavaScript, Java, Scala, Kotlin, VBScript, HTML5, T-SQL, PL/SQL	364 602	130	592	375	199	1 296	0.1/5.0

02

Scan Information

1/1 2024-09-04 09:36:07
11-SNAPSHOT.130256

Scan Statistics

Status
completed

Score
0.1/5.0

Duration
0:15:57

Lines of Code
364 602

Vulnerabilities

130	592	375	199
Critical	Medium	Low	Info

1 296
Total

Language	Status	Duration	Lines of Code	Number of Vulnerabilities				
				Critical	Medium	Low	Info	Total
Config files	completed	0:00:13	195 189	30	146	49	0	225
JavaScript	completed	0:13:10	127 615	8	88	178	132	406
JVM languages	completed	0:02:18	18 352	86	126	148	31	391
VBScript	completed	0:00:01	11 541	0	0	0	0	0
HTML5	completed	0:00:10	11 017	6	224	0	36	266
T-SQL	completed	0:00:01	446	0	0	0	0	0
PL/SQL	completed	0:00:01	442	0	8	0	0	8

Language Statistics

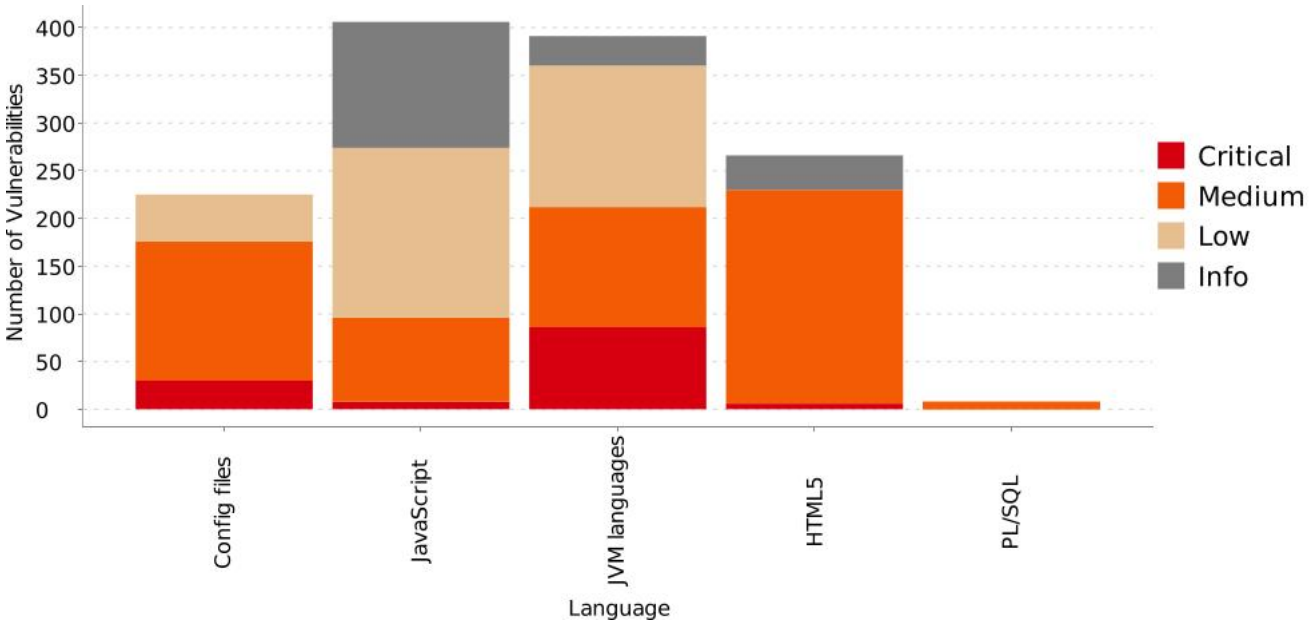
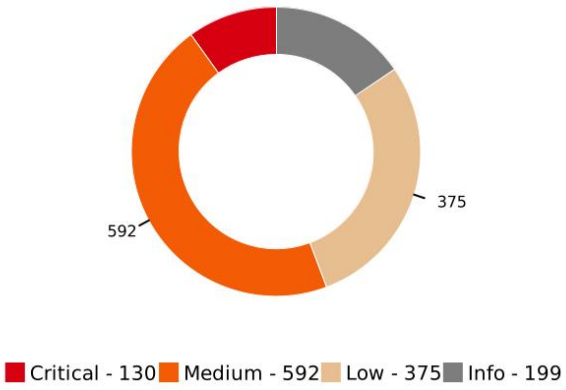
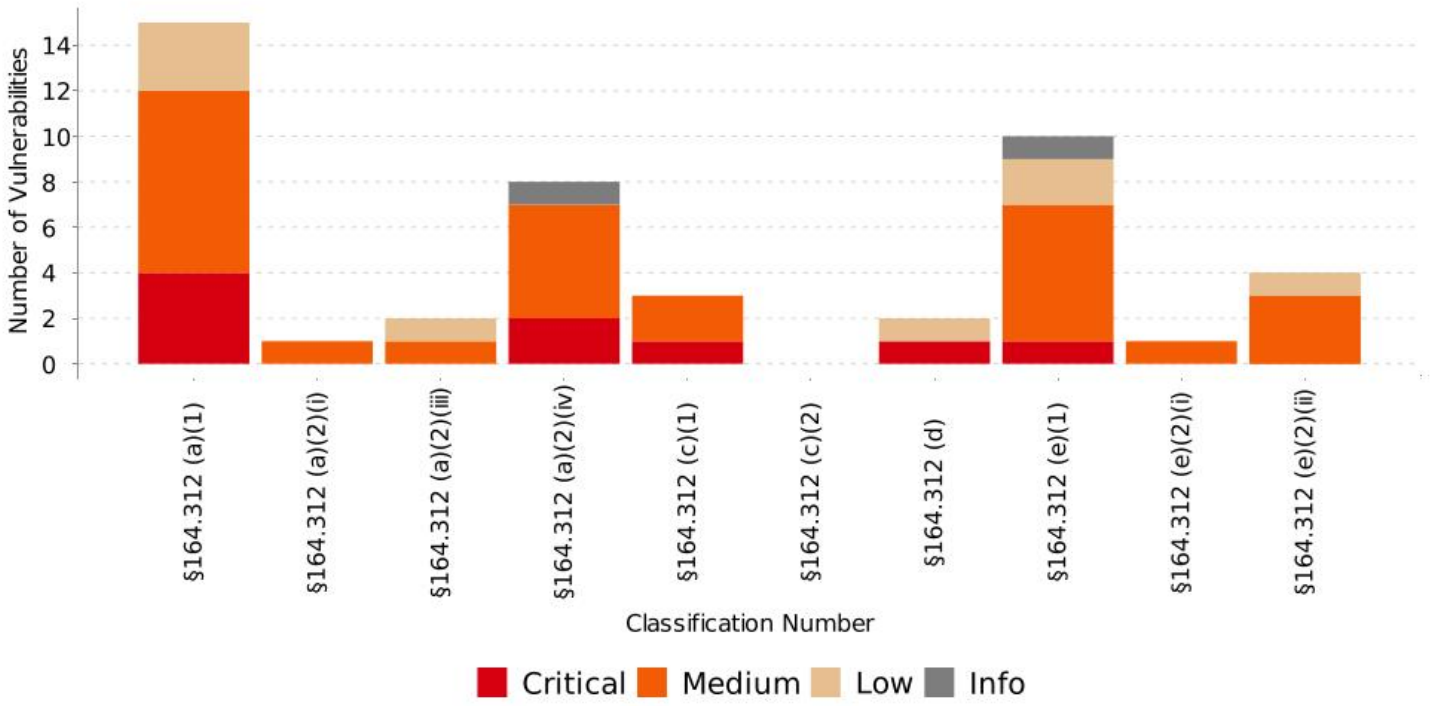


Diagram of identified vulnerabilities



Vulnerability Types

Classification by HIPAA



	Vulnerabilities					Occurrences				
	Critical	Medium	Low	Info	Total	Critical	Medium	Low	Info	Total
§164.312 (a)(1)	4	8	3	0	14	27	42	26	0	95
§164.312 (a)(2)(i)	0	1	0	0	1	0	8	0	0	8
§164.312 (a)(2)(iii)	0	1	1	0	2	0	58	8	0	66
§164.312 (a)(2)(iv)	2	5	0	1	7	15	28	0	1	44
§164.312 (c)(1)	1	2	0	0	3	6	7	0	0	13
§164.312 (c)(2)	0	0	0	0	0	0	0	0	0	0
§164.312 (d)	1	0	1	0	2	14	0	1	0	15
§164.312 (e)(1)	1	6	2	1	9	10	102	89	36	237
§164.312 (e)(2)(i)	0	1	0	0	1	0	6	0	0	6
§164.312 (e)(2)(ii)	0	3	1	0	3	0	87	49	0	136

Vulnerability List

Vulnerabilities are displayed accordingly to export settings: **4 selected**

Actual: **4 of 1296**

§164.312 (a)(1) — Access control

Critical vulnerabilities		**2***
SQL injection	Java	2
WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/advanced/SqlInjectionLesson6a.java:74		Confirmed
WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction/SqlInjectionLesson5b.java:86		Confirmed
Medium-level vulnerabilities		**2***
Path manipulation	Java	2
WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/MD5.java:49		Confirmed
WebGoat-main/src/main/java/org/owasp/webgoat/lessons/pathtraversal/ProfileUploadBase.java:42		Confirmed
Low-level vulnerabilities		0
Info-level vulnerabilities		0

§164.312 (a)(2)(i) — Unique user identification

Critical vulnerabilities	**0**
Medium-level vulnerabilities	**0**
Low-level vulnerabilities	0

* Rejected vulnerabilities are not taken into account

§164.312 (a)(2)(i)	Unique user identification
Info-level vulnerabilities	0

§164.312 (a)(2)(iii)	Automatic logoff
Critical vulnerabilities	0
Medium-level vulnerabilities	0
Low-level vulnerabilities	0
Info-level vulnerabilities	0

§164.312 (a)(2)(iv)	Encryption and decryption
Critical vulnerabilities	0
Medium-level vulnerabilities	0
Low-level vulnerabilities	0
Info-level vulnerabilities	0

§164.312 (c)(1)	Integrity
Critical vulnerabilities	0
Medium-level vulnerabilities	0
Low-level vulnerabilities	0
Info-level vulnerabilities	0

§164.312 (d)	Person or entity authentication
Critical vulnerabilities	2*

* Rejected vulnerabilities are not taken into account

§164.312 (d) **Person or entity authentication**

Critical vulnerabilities

SQL injection Java

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/advanced/SqlInjectionLesson6a.java:74	Confirmed
WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction/SqlInjectionLesson5b.java:86	Confirmed

Medium-level vulnerabilities	0
Low-level vulnerabilities	0
Info-level vulnerabilities	0

§164.312 (e)(1) **Transmission security**

Critical vulnerabilities	0
Medium-level vulnerabilities	0
Low-level vulnerabilities	0
Info-level vulnerabilities	0

§164.312 (e)(2)(i) **Integrity controls**

Critical vulnerabilities	0
Medium-level vulnerabilities	0
Low-level vulnerabilities	0
Info-level vulnerabilities	0

§164.312 (e)(2)(ii)

Encryption

Critical vulnerabilities	**0**
Medium-level vulnerabilities	**0**
Low-level vulnerabilities	**0**
Info-level vulnerabilities	**0**

Analysis Results

§164.312 (a)(1)

§164.312 (d)

SQL injection (Java)

Description

SQL injection is possible. This can be exploited to bypass the authentication mechanism, access all database entries, or execute malicious code with application rights.
Client side code injection attacks take the first place in the “OWASP Top 10 2017” web application vulnerabilities ranking and the seventh place in the “OWASP Mobile Top 10 2014”. ranking. The level of potential damage from such an attack depends on the user input validation performance and file protection mechanisms.
SQL Injection occurs when a database query is based on data from an untrusted source (e.g., user input). In the absence of proper validation an attacker can modify the query to execute malicious SQL query.
The most common variants of SQL injection:

- Direct addition of malicious code into a string variable, based on which the SQL query is generated.
- Premature termination of the correct SQL command via the “– ” sequence of characters (interpreted as the beginning of a comment). The contents of the string after this sequence will be ignored during the execution of SQL command.
- Addition of the “;” character (interpreted as the end of the command), and other malicious commands (request splitting) to the input string variable.
- Password guessing via the sequential execution of SQL queries.

Example

Let the application substitute the user entered city name to the SQL query. For example, for the input string “London” we have:
SELECT * FROM Orders WHERE City = 'London'
An attacker can enter the following query:
London'; drop table Orders--
In this case, a query will be built that searches through the table and then deletes the table:
SELECT * FROM Orders WHERE City = 'London';drop table Orders-- '

Recommendations

- Do not make assumptions about the type or amount of data entered by a user.
- Implement a mechanism of validation for data entered by a user.
- Escape special characters (“;”, “–”, “/*”, “*/”, “‘”; the exact list depends on the database type).
- Use stored procedures to validate user input along with the mechanism of parameters filtering.

Links

1. OWASP Top 10 2017-A1-Injection
2. OWASP: SQL Injection
3. WASC-19: SQL Injection
4. CAPEC-66: SQL Injection
5. Understanding SQL Injection – cisco.com
6. CWE CATEGORY: OWASP Top Ten 2017 Category A1 - Injection
7. CWE-89

Vulnerability Entries

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/advanced/SqlInjectionLesson6a.java:74

Level Critical

Status Confirmed

```
71 try (Statement statement =
72    connection.createStatement(
73       ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY)) {

74   ResultSet results = statement.executeQuery(query);

75
76   if ((results != null) && results.first()) {
77    ResultSetMetaData resultsMetaData = results.getMetaData();
```

Trace

@org.springframework.web.bind.annotation.PostMapping("/SqlInjectionAdvanced/attack6a")
@org.springframework.web.bind.annotation.ResponseBody
public org.owasp.webgoat.container.assignments.AttackResult completed(@org.springframework.web.bind.annotation.RequestParam("userid_6a") java.lang.String userId)

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/advanced/SqlInjectionLesson6a.java:56#60

```
53
54 @PostMapping("/SqlInjectionAdvanced/attack6a")
55 @ResponseBody
56 public AttackResult completed(@RequestParam(value = "userid_6a") String userId) {
57   return injectableQuery(userId);
58   // The answer: Smith' union select userid,user_name, password,cookie,cookie, cookie, userid from
59   // user_system_data --
60 }
61
62 public AttackResult injectableQuery(String accountName) {
63   String query = "";
```

Statement.executeQuery()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/advanced/SqlInjectionLesson6a.java:74

```
71 try (Statement statement =
72     connection.createStatement(
73         ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY)) {
74   ResultSet results = statement.executeQuery(query);
75
76   if ((results != null) && results.first()) {
77     ResultSetMetaData resultsMetaData = results.getMetaData();
```

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction/SqlInjectionLesson5b.java:86

Level Critical

Status Confirmed

```
83 // String query = "SELECT * FROM user_data WHERE Login_Count = " + login_count + " and userid
84 // = " + accountName, ;
85 try {
86   ResultSet results = query.executeQuery();
87
88   if ((results != null) && (results.first() == true)) {
89     ResultSetMetaData resultsMetaData = results.getMetaData();
```

Trace

Connection.prepareStatement()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction/SqlInjectionLesson5b.java:65

```
62 String queryString = "SELECT * From user_data WHERE Login_Count = ? and userid= " +
accountName;
63 try (Connection connection = dataSource.getConnection()) {
64   PreparedStatement query =
65       connection.prepareStatement(
66           queryString, ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.
CONCUR_READ_ONLY);
67
68   int count = 0;
```

PreparedStatement.executeQuery()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/sqlinjection/introduction/SqlInjectionLesson5b.java:86

```
83 // String query = "SELECT * FROM user_data WHERE Login_Count = " + login_count + " and
userid
84 // = " + accountName, ;
85 try {
86   ResultSet results = query.executeQuery();
87
88   if ((results != null) && (results.first() == true)) {
89     ResultSetMetaData resultsMetaData = results.getMetaData();
```

§164.312 (a)(1)

Path manipulation (Java)

Description

Using data from an untrusted source when working with the file system may give an attacker access to important system files.
By manipulating variables that reference files with “dot-dot-slash (../)” sequences and its variations or by using absolute file paths, it may be possible to access arbitrary files and directories stored on file system including application source code or configuration and critical system files.

Example

In the following example, the application uses the value of the HTTP request parameter to specify the name of the file that is to be deleted. An attacker can set the string ../../tomcat/conf/server.xml as a parameter and thus delete the configuration file.
String rName = request.getParameter("reportName");
File rFile = new File("/usr/local/apfr/reports/" + rName);
rFile.delete()

Recommendations

- Create a white list of acceptable names from which the user can choose. Do not use values entered by the user without validation.

Links

1. OWASP Top 10 2017-A1-Injection
2. OWASP Top 10 2017-A5-Broken Access Control
3. OWASP Top 10 2013-A4-Insecure Direct Object References
4. CWE-73: External Control of File Name or Path
5. Path Traversal - OWASP
6. CWE CATEGORY: OWASP Top Ten 2017 Category A1 - Injection
7. CWE-23
8. CWE-36
9. Restrict path access to prevent path traversal
10. A01:2021 - Broken Access Control
11. CWE-35: Path Traversal
12. A03:2021 - Injection

Vulnerability Entries

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/MD5.java:49

Level Medium

Status Confirmed

```
46 } else {
47   for (String element : args) {
48     try {
49       System.out.println(MD5.getHashString(new File(element)) + " " + element);
50     } catch (IOException x) {
51       System.err.println(x.getMessage());
52     }
```

Trace

```
/** * Command line program that will take files as arguments and output the MD5 sum for each file. * * @param args * command line arguments * @since ostermillerutils 1.00.00 */
public static void main(java.lang.String[] args)
```

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/MD5.java:43#55

```
40  * @param args command line arguments
41  * @since ostermillerutils 1.00.00
42  */

43 public static void main(String[] args) {
44   if (args.length == 0) {
45     System.err.println("Please specify a file.");
46   } else {
47 ...
48       }
49     }
50   }
51 }

52
53 /**
54  * Gets this hash sum as an array of 16 bytes.
```

new File()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/challenges/challenge7/MD5.java:49

```
46 } else {
47   for (String element : args) {
48     try {

49       System.out.println(MD5.getHashString(new File(element)) + " " + element);

50     } catch (IOException x) {
51       System.err.println(x.getMessage());
52     }
```

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/pathtraversal/ProfileUploadBase.java:42

Level Medium

Status Confirmed

```
39 File uploadDirectory = cleanupAndCreateDirectoryForUser();
40
41 try {
42   var uploadedFile = new File(uploadDirectory, fullName);
43   uploadedFile.createNewFile();
44   FileCopyUtils.copy(file.getBytes(), uploadedFile);
45
```

Trace

@org.springframework.web.bind.annotation.PostMapping(value

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/pathtraversal/ProfileUpload.java:36#40

```
33     consumes = ALL_VALUE,
34     produces = APPLICATION_JSON_VALUE)
35 @ResponseBody
36 public AttackResult uploadFileHandler(
37     @RequestParam("uploadedFile") MultipartFile file,
38     @RequestParam(value = "fullName", required = false) String fullName) {
39   return super.execute(file, fullName);
40 }
41
42 @GetMapping("/PathTraversal/profile-picture")
43 @ResponseBody
```

new File()

WebGoat-main/src/main/java/org/owasp/webgoat/lessons/pathtraversal/ProfileUploadBase.java:42

```
39 File uploadDirectory = cleanupAndCreateDirectoryForUser();
40
41 try {
42   var uploadedFile = new File(uploadDirectory, fullName);
43   uploadedFile.createNewFile();
44   FileCopyUtils.copy(file.getBytes(), uploadedFile);
```

45

03

About HIPAA

The report contains vulnerability classification by **HIPAA** (Health Insurance Portability and Accountability Act). HIPAA defines a set of privacy (HIPAA Privacy Rule) and security (HIPAA Security Rule) standards for health information. The HIPAA standards apply to medical organizations and insurance companies. From the set of standards, those that can be checked using the system were selected.

Note that some vulnerabilities may belong to the number of categories or to none at all. To see the full list of vulnerabilities, choose the **By severity** classification method.

§164.312 (a)(1)

Access control

Implement technical policies and procedures for electronic information systems that maintain electronic protected health information to allow access only to those persons or software programs that have been granted access rights

§164.312 (a)(2)(i)

Unique user identification

Assign a unique name and/or number for identifying and tracking user identity

§164.312 (a)(2)(iii)

Automatic logoff

Implement electronic procedures that terminate an electronic session after a predetermined time of inactivity

§164.312 (a)(2)(iv)

Encryption and decryption

Implement a mechanism to encrypt and decrypt electronic protected health information

§164.312 (c)(1)

Integrity

Implement policies and procedures to protect electronic protected health information from improper alteration or destruction

§164.312 (c)(2)

Mechanism to authenticate electronic protected health information

Implement electronic mechanisms to corroborate that electronic protected health information has not been altered or destroyed in an unauthorized manner

§164.312 (d)

Person or entity authentication

Implement procedures to verify that a person or entity seeking access to electronic protected health information is the one claimed

§164.312 (e)(1)

Transmission security

Implement technical security measures to guard against unauthorized access to electronic protected health information that is being transmitted over an electronic communications network

§164.312 (e)(2)(i)

Integrity controls

Implement security measures to ensure that electronically transmitted electronic protected health information is not improperly modified without detection until disposed of

§164.312 (e)(2)(ii)

Encryption

Implement a mechanism to encrypt electronic protected health information whenever deemed appropriate