GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials: Network, Endpoint, and Cloud (Security 401)"
at http://www.giac.org/registration/gsec

Security Testing of web applications : Best Practices and Tools

© SANS Institute 2005, Author retains full rights.

Author : Sridhar Ponnapalli
Date Submitted : November 14, 2004
GSEC Practical Assignment
Version v.1.4c
Option 1

© SANS Institute 2005 Author retains full rights.

1. Abstract

In today's increasingly inter-networked world, businesses heavily depend on web-based applications to conduct their business, and bring revenue. Applications like company web sites and portals, CRM and sales sites, web services, and EDI have become the backbones for successful running of a modern organization. As sophisticated as today's applications are, the hackers or the "Bad guys" have become even more sophisticated to exploit any possible application vulnerabilities, to compromise company's resources and to damage company's ability to conduct business. In the lifecycle of software development, the architects and designers may come up with a secure architecture and design, developers may write secure code, but if security is not made a part of testing, a functional but vulnerable system will be rolled out, with a potential to cause huge revenue losses.

This paper proposes best testing practices and useful tools for security testing of web applications against various vulnerabilities, and within different components of web applications, that can wreak havoc on the critical infrastructure of a company. It also refers tools that can be handy in performing security testing efficiently. "web applications" for the purpose of this paper include any intranet, extranet or internet applications built within an organization, that face internal or external customers. The targeted audiences for this paper are testing or Quality Assurance team, developers, security professionals, or any IT professional from an organization that participates in security testing of web applications.

2. Security testing – A neglected child

Most corporations usually make significant investments in robust network security solutions, like firewalls, IDS (Intrusion Detection Systems), IPS (Intrusion Prevention Systems). Most organizations also invest precious resources in various essential security efforts like risk assessments, certification, security architectures and policies, penetration tests etc, but are usually tempted to ignore or give low priority to thoroughly testing web applications for security holes, prior to deployment. While general or network level security tests bring out gaps like application server patch levels, vulnerabilities in third party software etc, they are not aimed at revealing application layer security flaws in home grown web applications. Organizations need to understand that vulnerable applications can be a hacker's gateway into private networks. It is an established fact that dealing with a major or even a minor attack after it hits, can be very expensive and can severely undermine the reputation of an organization. This makes it imperative that organizations mandate security testing without exceptions, for all of their web applications prior to deployment, and make security testing an integral part of Software Development Lifecycle, from a very early stage.

© SANS Institute 2005 Author retains full rights.

3. Attack Types and Testing Recommendations

This section discusses some of the most common threats web-based applications face in today's world, and provides testing recommendations, for each threat. Each sub-sections also refers tools that can be used to test web applications for specific vulnerabilities, and some of the referred tools are discussed in detail in section 6 *Useful tools for web application security testing*.

This section is not intended to be a comprehensive list of threats for web-based applications, and threats/vulnerabilities that are not related to web applications (like network vulnerabilities etc) are beyond the scope of this discussion.

3.1 Buffer overflow

"Buffer overflows account for nearly 80% of the security vulnerabilities reported in software" (Whittaker & Thompson, p.171)

Problem : Buffer Overflow occurs when larger input data is provided to a program than it is prepared to hold. For example, if a string is declared to be of size 20 in the application, and a 40 character input is provided to that, this may result in a buffer overflow. The excess data overflows or spills into adjacent buffers, which is the program area, causing the application to surrender control to the user. An attacker may use this cleverly to insert his/her own covert commands into the program stack, by making them a part of the large input provided, and run them on the host server. Once an attacker is able to run one or more rogue processes on the host server, in most cases, he/she can get root or Admin rights on the host server, which can lead to catastrophic consequences. Even if the attacker is not able to run malicious code on the host server, harm can still be done if the attacker can make the system hang by repeated invocations using large input values. This can lead to Denial of Service attack, which is discussed later in the document. Though buffer overflows are more common in applications built using C/C++ programming languages because of a lack of inherent bounds checking on variables and arrays, they are nevertheless possible with other programming languages too (like Java, VB, Perl etc).

Testing Recommendations :

- Testing team needs to understand the boundary conditions for each input field. Application developers should provide the testers with the size, type and format of each input field along with the expected behavior when boundary conditions are violated on the field. Testers should then prepare test cases based on this information and run those test cases to check the behavior of the application when boundary conditions are met or exceeded.

© SANS Institute 2005 Author retains full rights.

- Testers should run test cases for each input value, where the type of data differs from the type the application is expecting. For example, making sure the application gracefully handles situations in which an alphabet is passed for input where the application is expecting a numeric.
- As Buffer Overflow attacks are platform specific (dependent on hardware, OS, system software), chances of attack can be reduced by denying an attacker the platform-related knowledge. Negative testing needs to be performed to make sure the application is not giving away platform related information, for malicious requests. This topic is covered in more detail in the sub-section *Improper error handling*.
- Testing for buffer overflow should include making sure that any overflows are detected and handled by the application at a very early stage in the process, and not after quite a bit of processing. Bounds-checking for input variables should be done by client-side code or at least, at the very beginning of server-side code, rather than at the database level, after already running some business logic at the application-level.
- Testers are encouraged to test the application by inputting different types of Unicode characters like Japanese and Chinese characters, by copying these from other web sites, as sometimes Buffer overflow can be caused by feeding such characters to the application.

Several code scanning tools are available in the market that can be used to detect buffer overflows in application code. Cenzic Hailstorm and Watchfire AppScan are two such tools, and both these are discussed more in section 6.

For more information regarding buffer overflow attacks and countermeasures, please refer to the following articles.
http://www-106.ibm.com/developerworks/security/library/s-overflows/
http://www.rsasecurity.com/rsalabs/node.asp?id=2011

3.2 Denial of service (DoS)

Problem : Denial of service is an attack on a web site or service, inundating it with high number of malicious requests to consume all of the system or network resources, to make the site/service unavailable to legitimate users. For example, a hacker may generate thousands of automated requests to a web site that provides free e-mail service on the internet, filling up available space, and shutting down the service. Denial of service attacks can lead to lost revenue, and impaired customer confidence. The most common target resources for this attack include memory, network bandwidth, application connections, database connections, CPU or disk space. In case of web applications that haven't been load-tested, a denial of service scenario can occur even without any malicious attempts, because of lack of available resources to handle all legitimate users, i.e. many innocent concurrent users can bring the system/service down.

© SANS Institute 2005 Author retains full rights.

Denial of Service may sound as an entirely network related vulnerability, but is discussed in this paper, as applications have a good role to play in being designed, developed and tested to be resistant to these types of attacks. So, only the application side of DoS attacks is covered here, and discussion of pure network type of DoS attacks like port flooding or SYN flooding, and of the tools to prevent or detect such attacks, is beyond the scope of this paper.

Testing Recommendations :

- Implementing client session timeouts and releasing associated resources is an important factor in securing a web application against DoS attacks. Without client timeouts, the application can reach its thresholds faster, as it keeps all sessions open, consuming valuable resources. This should be tested thoroughly by the testing team to make sure client sessions are being timed out, and resources are being released in a timely manner.
- Testers should understand from design, the exhaustion limits on resources such as memory, CPU, disk, database connections or any other application resources. Testing should include simulating load for the expected number of maximum concurrent users, effectively meeting or exceeding resource limits to validate the application behavior in these cases. It may not be trivial to simulate high loads, but should be possible, using automated tools, as referred subsequently in this section
- Testers should check if the application is enforcing user-level thresholds as against global thresholds, wherever possible. User-level thresholds are better because they make it difficult or sometimes impossible for a hacker to consume other users' resources. For example, by setting a database parameter that determines the maximum number of concurrent database connections that can be open per user, a hacker can only fill up his connections, but cannot consume connections from other legitimate users.
- When errors occur in a web application, test to verify that application exits only after completing all housekeeping tasks. For example, closing any open files, open database connections, database cursors opened by application etc. If not, this increases the likelihood of accidental denial of service, without any malicious attempts
- Ensure that redundancy of service is implemented in the application, but keep in mind that if network bandwidth is choked as a result of DoS attack, failover mechanisms will need a whole separate redundant network for business continuity. Testing redundancy can be accomplished by bringing down the components of the web application one at a time, and checking the application response.
- Testing team should check the verbosity of the logs generated, which is usually made a configurable parameter (turn on/off, or different levels). If verbose output is always on, or if certain modules of the application generate extremely verbose log entries, an attacker may exploit this, to fill out the disk space very quickly

© SANS Institute 2005 Author retains full rights.

- As mentioned in the CERT article referenced below, testing should include verifying that emergency procedures are defined and understood, to handle situations where a DoS attack disables most or all of the privileged logins of a server, through failed login attempts.
- It is increasingly becoming a common practice to incorporate anti-automation techniques in web sites and applications, to prevent DoS attacks, which almost always need automation to succeed. Anti-automation is a technique that prevents automated programs from exercising the functionality of a web site, service or application, by having a test that only a human would pass. For example, when you try to create a new account with Yahoo, you are required to enter the code from an image displayed, failing which you won't be able to register. If implemented, anti-automation techniques should be tested extensively by testing team, by trying to invoke the application through scripts or programs.

Load testing tools like Mercury LoadRunner (http://www.mercury.com/us/products/performance-center/loadrunner/), or Empirix e-Load (http://www.empirix.com/Empirix/Web+Test+Monitoring/testing+solutions/web+application+load+testing.html) can be used to simulate high enough user load to meet or exceed boundary conditions, to verify the application behavior under high load.

For more information on Denial of Service vulnerability, please refer to the following CERT article
http://www.cert.org/tech_tips/denial_of_service.html

3.3 Improper error handling

Problem : Every application handles error conditions and throws out error messages. If the error messages from a web-based application are too detailed, this may make the application vulnerable, as hackers will attempt to learn valuable system related information, which will mean a high "Hit Rate" for an attack. For example, returning the source code of a failed SQL statement, name or version of the web or application server, or operating system version on which the application is running, will provide an intruder with crucial information that can increase the likelihood of a successful attack. Denying sufficient knowledge of the operating platform of a web application handicaps an attacker attempting almost any type of attack on applications.

Testing Recommendations :

- Development team should prepare and pass on to the testing team, a comprehensive list of error conditions along with the detailed messages output by the application, as a part of test entrance criteria. This list should contain not only functional errors, but also internal conditions like database or web server being down, network errors, out of memory errors etc. Testers should work with development and other teams as required, to create these scenarios for testing.

© SANS Institute 2005 Author retains full rights.

- Application should be tested for all types of error messages, even the ones that may not be documented in the design documents, like null pointer exceptions, core dumps, failed database connections etc. There is no definite checklist for this type of testing, except trying arbitrary input values, URL strings, parameters etc with the application, preferably ones that include non alphanumeric characters, and validating that the system messages are generic enough.
- Testing team should validate that web applications are following a principle of most generic error message. For example, applications should provide messages like *problem processing request. Please contact…*, wherever possible, while providing more detailed diagnostic information in the log files, to help troubleshooting.
- Sometimes, detailed messages divulging potentially harmful system information are found not on the web page that appears, but on a linked page or in the HTTP header. Testing team should check all links, as a part of negative testing, to verify such gaps are not left unnoticed. Testing team should also work with developers to dump HTTP request and response headers into text files, to ensure they don't contain error messages revealing sensitive information.
- It is a very common practice for an application to have debug messages providing a lot of information, during the build stage. Testing should validate that all the debug messages are removed from the application, prior to deployment.

Cenzic Hailstorm is a tool presented in section 6, and can be used to detect improper error handling vulnerability in application code. Alternatively, generic test scripting tools like *eValid* from Software Research Inc (www.soft.com) can be used to create scripts to generate all kinds of errors from the application with various input values, and validate resulting messages and HTML for any leakage of confidential information

For more information on improper error handling vulnerability, please refer to the following OWASP article
http://www.owasp.org/documentation/topten/a7.html

3.4 SQL injection attack

Problem : SQL injection attack occurs when a hacker "injects" malicious code into dynamic SQLs through web forms, to be queried directly against the database. A successful SQL injection attack exploits code that doesn't properly filter input, and can inflict quite a bit of damage, like destroying the database or bringing down the web application. To study how SQL injection works, consider the following piece of code, used to build a dynamic SQL

myQuery = "select username from users where username = '" & inUser & "' and password = '" & inPassword & "'"

Normally, if you enter a valid login/password, this code builds a query like the following, to be run against the database.

© SANS Institute 2005 Author retains full rights.

select username from users where username = ‘abc’ and password = ‘xyz’

If the user enters *abc‘ or 2 > 1--* into *login* field, the SQL gets changed to

select username from users where username = ‘abc’ or 2 > 1 -- …..

‘--‘ comments out the rest of line in the query in some database servers (for example, SQL server). In this context, the last query above will be always true because of the universally true predicate *2 > 1* being OR’ed with any other condition, and may bypass authentication to allow access to the web site or the database, even if a blank or wrong password is provided. Once an attacker is able to run arbitrary queries at will, in most databases, he/she can then run special database procedures that grant them command execution privilege at OS level.

Testing Recommendations :

- Testing by inputting special characters should be made a part of the test cases to validate that the application is handling known attack methods, and denying access to an attacker. Try typing characters like -- (double-hyphen), ; (semi colon), ‘ (single quote), “ (double quote) and other similar characters of special meaning in different flavors of SQL (Oracle, MS SQL server etc), into input fields of the application. The next step could be to proceed with appending universally true expressions like the one mentioned above, at the end of all valid input fields using different expressions coupled with a variety of SQL keywords like AND, OR, UNION etc, to simulate an attack. SQL injection attacks can also happen by injecting malicious code into the URL. For example, by adding *?userid=xyz’ or 2 >1--* at the end of a URL that is expecting userid as the first parameter, may give away access to the user, without a valid password
- Testers are encouraged to work with developers and DBAs to find out if the application needs special system or built-in database procedures that can grant command execution privilege at OS level, for its functionality. If the application doesn’t need it, it is better to delete such procedures from the database, to help enforce principle of least privilege and reduce exposure, in the event a hacker breaks in.
- Testing should include validating that all web database users for the application have limited privileges, only to perform the required transactions. It is advised to create a separate user or role for web users, because their privilege levels are different from application developers, DBAs or internal users. This can help reduce the damage when an attacker breaks in.
- In case of SQL injection attacks, passwords stored in clear text in the database can drastically expand the scope of attack, as once the attacker is in, he/she can gather passwords for a variety of systems/applications to inflict damage. Testers are encouraged to sweep the tables in the application schema, especially the ones with indicative names like USERS, USER_PROFILE, USER_INFO etc, to ensure that sensitive information is not readily available.

© SANS Institute 2005 Author retains full rights.

Some of the useful tools for testing web application code for SQL injection vulnerability are :

- SPI toolkit from SPI Dynamics (http://www.spidynamics.com/products/Comp_Audit/toolkit/index.html)
- Spike proxy from Immunity (http://www.immunitysec.com/resources-freesoftware.shtml)

For more information on SQL injection vulnerability, please refer to the following white paper
http://www.spidynamics.com/whitepapers/WhitepaperSQLInjection.pdf

3.5 Cross site scripting (XSS)

Problem : Cross site scripting is an attack that involves web sites inadvertently including malicious HTML tags or script in a dynamically generated page based on unvalidated input from untrustworthy sources (CERT2). The hacker sends bait usually in the form of an attractive link on a web site, sent in an e-mail, or as a malicious message posted on a discussion group. When the unsuspecting user clicks the link, response sent from the host contains malicious commands within scripting tags like <SCRIPT>, <OBJECT>, <APPLET>, <EMBED> etc, which are run on the user's computer. This can result in a variety of threats, including cookie harvesting, session hijacking, impersonation etc. Kevin Spett (Spett2) mentions the following in his article *Cross-Site Scripting. Are your web applications vulnerable ?*

> Cross site scripting can potentially impact any site that allows users to enter data like :
>
> - Search engines that echo the search keyword that was entered
> - Error messages that echo the string that contained the error
> - Forms that are filled out which later present the data entered to the user
> - Web message boards that allow users to post their own messages

Testing Recommendations :

As XSS attacks are triggered by a vulnerable application and client-side actions, care should be taken on both client and server sides to minimize the likelihood of these attacks

© SANS Institute 2005 Author retains full rights.

- Web applications should thoroughly validate all dynamically generated output HTML, using inclusion logic, rather then exclusion logic. This means that data used to generate dynamic HTML will be validated against a set of valid characters expected for each field, and will reject any value that doesn't meet this criteria. This will help filter out special characters like <, > etc, which are used by attackers in scripting tags, thereby minimizing the exposure to such attacks. Though extensive field-level validation can belabor coding and can adversely impact performance to some extent, this is still advised, as it provides an excellent security net for web applications
- One workaround from a client side is to disable scripting languages like Javascript, VB script, ActiveX or Java applet in the browsers of each user. This is considered by many as the most effective solution against XSS attacks, but has the potential to render many web sites unusable, as quite a few sites use client-side scripts heavily for their functionality. If this is the approach taken, it needs to be tested extensively by the testing team by disabling scripts or active content in the client, to confirm that critical functionality of the application is not impaired for clients that disable scripts.

As a preventive measure (not for testing), users should be educated about the ramifications of promiscuous browsing, and should be advised not to click on untrusted links in web sites, discussion groups, emails etc. This should be in line with the security policies of most corporations.

There are several good tools available to assist security testers in testing applications for cross site scripting vulnerability, like Cenzic Hailstorm and AppScan discussed in the section *Useful tools for web application security testing*, as well as tools like KSES (http://sourceforge.net/projects/kses)

For more information on Cross site scripting vulnerability, please refer to the following white paper
http://www.spidynamics.com/whitepapers/SPIcross-sitescripting.pdf

3.6 Privilege Elevation

Problem : Privilege Elevation vulnerability can be defined as the ability of a user with no or restricted privileges on an application or network, to illegally gain unauthorized higher-level privileges. The risk of Privilege Elevation attacks is high partly due to the fact that most of today's web or application servers require their processes to be run as root (for Unix servers) or administrator (for Windows servers). Once an attacker breaks in, he/she is normally able to get server level access with the privileges of the process that was used to break in. This obviously gives the hacker complete access of the host server, and hence has potential for irreparable damage.

© SANS Institute 2005 Author retains full rights.

Testing Recommendations :

- Testers should check if the application enforces the principle of least privilege. For example, does the application easily allow creation of user groups with different access roles ? This not only makes it easy for the administrators to have tighter controls on the privilege levels, but also makes it non-trivial for attackers to elevate privilege levels at will, even if they break into the network. Testing should ensure that :
    a) All privilege levels are properly defined in the system requirements, before starting the testing
    b) Each privilege level defined within the system requirements is implemented accurately, i.e. can perform only the set of transactions that it is entitled to, and does not have any privileges to perform transactions of a higher privilege level. Testers can test this by trying to perform each transaction with one less privilege level than the role or group that is entitled for the transaction and check if it goes through.
- Testers are encouraged to check the URL immediately after logging into a web application. Sometimes applications use constructs like *usertype=user*, as a part of the URL. In such cases, a good way of negative testing is to change this to *usertype=admin*, or *usertype=root* and see if they can get more privileges just by tampering the URL.
- Web applications should implement a proactive warning system, to send out automatic notifications if a job that usually gets executed by a normal user, gets kicked off by an Administrator. If implemented, testing should verify that this functionality is working as expected.
- Are all the off-the-shelf components like database, web server, browser etc, used by the application kept up to date with security patches ? This is very critical for the overall security of a web application, and should be verified by the testing team, prior to deployment.

The tests recommended above for privilege elevation are best done manually. However, in case of very large enterprise systems, general test script automation tools like Empirix e-Test suite (http://www.empirix.com/Empirix/Web+Test+Monitoring/Testing+Solutions/Integrated+Web+Testing.html) can be used to create automated scripts to test for privilege elevation vulnerability.

Please refer to the article http://www.watchguard.com/infocenter/editorial/135144.asp for more information on privilege elevation attacks

© SANS Institute 2005 Author retains full rights.

4 Best practices for testing application components

Every web application is built on several architectural components. The components of a typical web application include the client (browser), web server, application server, and the database server. Each of these components serves a specific purpose, faces different security challenges, and needs to be tested specifically for that component's functionality. The previous section discussed how to test applications for some of the most prevalent application-level security vulnerabilities that exist today. However, security testing of web applications is not complete by only testing for vulnerabilities, and should necessarily include testing of the individual components. This section attempts to provide best practices for security testing of each of the application components.

4.1 Client Security

Client security is a very important part of the overall functionality of a web application. Following are some best practices that specifically target ensuring security of the client component of web applications. Note that a web application is only as secure as its most insecure link, and don't let that be the application's client!

- If the application uses cookies to implement stateful connections, verify that the application uses SSL to encrypt the username, password and any other sensitive information in all communications, and that cookies are stored encrypted on the client. Sometimes replay attacks may be executed by copying contents of stolen cookies (like userid, password, session id etc), either in encrypted or unencrypted form into URLs, or HTML forms. Testers are encouraged to try this technique, in an attempt to simulate replay attacks.
- Testing team should test the navigation of the web application thoroughly, to verify visitors cannot jump all over the place to breach the navigational integrity of the application. For example, an e-commerce application that allows users to go directly to the order submit page, without going through the payment and shipping pages, through some combination of Go, history, browser's back button etc, can cause serious data integrity problems on the backend, and opens the web site or application for illegal use (Splaine, p. 118).
- If the client machine crashes in the middle of an application run, it may leave sensitive information stored in cookies or temporary (session-level) files on the hard disk of the user (Splaine, p.122). Processes should be setup to periodically check into the directories or locations where application stores temporary data, to make sure these areas are sanitized, and leave no clues for an attacker.

© SANS Institute 2005 Author retains full rights.

- Testing should ensure that the application functions well with all supported browser types, browser versions, and client-side security settings. This testing gets easy in corporate environments where all clients are homogeneous, but gets to be more difficult in public places like libraries, cyber cafes etc, where client standards cannot be enforced. The application may not be fully functional or secure, unless rigorous testing is performed with all the permutations and combinations of the client settings. Testing should also include checking if supported browser types and versions are clearly mentioned at a very visible location on the web site (typically homepage of the application)
- If clients invoke the web application from behind a proxy server, this can cause some information (like source IP address) to be masked by the proxy server, that is required by the application (Splaine, p.113). Testing should include verifying that design of the application addresses this scenario. For example, if a module of the application requires the source IP address, that may not function as expected in this context. Other similar scenarios like client accessing from behind a firewall, client with content from multiple domains disabled, or server using reverse proxy should also be included in testing
- Signing code is a suggested best practice for web application integrity. Code signing or content signing involves generating a unique one-way hash of content that is being signed, and assures that code hasn't been tampered while being transmitted over the network. Though code signing can be done for client or server code, signing client components like ActiveX, Javascript, Applets etc is considered more critical. This is because of the higher likelihood that client components may be tampered as they are traversing through the internet, than the server components, which reside and run on the server side. If code signing is implemented, testing should ensure that signed code can run on all the client machines without any issues.

4.2 Web server/Application server security

However strong an application's client security is, or however robust an application's design is, needless to say that the application as a whole will be vulnerable if the web server and the application server are not secure enough. Following defense-in-depth approach, it is very essential to make sure that web server and application server components are as secure as they can be, especially given the possibility that attacks may sometimes bypass the client component. Following are some best practices to verify that this very important aspect of web applications is secure.

© SANS Institute 2005 Author retains full rights.

- Validating input values only on the client side leaves a big security hole in a web application. Attackers can easily generate HTTP requests that bypass client side logic, or there may even be a genuine need for an application to be invoked directly from the backend. For example, a web application may have a daily process that invokes application through a backend script, to repost transactions that have failed during the day. Having some basic validation on the client side may still be useful for performance, so that the number of invalid requests sent to the server is minimized. But it is critical to validate all input values on the server side, to make sure any requests bypassing the client logic don't make their way through the application logic.
- It is very important to validate that any files or resources on the same host as the web server do not contain sensitive information. Is there a need for all the files and information to be present on the same physical server as the web server ? If not, move any files or sources of confidential information to separate servers, including log files, configuration files etc.
- What does the application record about transactions ? Are logins, passwords being written to log files in clear text ? Testing should include these checks, because such security holes pose a threat not only from outside hackers, but from insiders too.
- Test to ensure that all sensitive data used or created by the application server is encrypted, and the keys are securely stored. Data like IP addresses, user accounts, passwords, customer data, lists of prospects or leads, credit card numbers etc should never be stored in plain text either in the database or on the file system. This helps restrict damage level in the event of a successful attack.
- How does the application handle expired or revoked certificates ? The preferred way of handling is to warn the user a few days/weeks before the certificate expires, and when the certificate expires, to stop execution with a clear error message, without executing any application logic. Testers need to validate that the application is doing this at the very beginning of the process, and not stopping in the middle, causing integrity problems. Also, if not properly tested, application may give away sensitive information in error messages when certificates expire or are revoked, causing improper error handling attacks. Expired certificates for testing can be obtained from third-party certificate vendors, or created by self-signing.
- Are all the services and features which are turned on, required for the application to function ? Remember that every extra service/port open, could potentially provide an additional access point for a hacker, and turning off unnecessary services/ports tremendously helps reduce the security exposure of an application. Testers are encouraged to get a list of all ports that are open for the application, and ensure each of the open ports supports some functionality specified in the functional or system requirements, and suggest removal of any unused ports.

© SANS Institute 2005 Author retains full rights.

4.3 Database Security

Protecting a database from attacks is very critical to the overall security posture of a web application. In most real life web applications, databases store very sensitive information, like user profiles, customer details etc, in addition to the application data. Hence if the database is not secure enough, it leaves the security infrastructure of the application incomplete and vulnerable. Following are some guidelines to testing database security of a web application.

- It is a highly advocated and safe practice to have the database server behind a firewall, and configured to allow requests to come in only from trusted hosts or IP addresses (like web server, application server etc). Though this doesn't make the database server impregnable (as attackers can still inject code through the web site), it will limit the points of exposure of the application from external, and internal users. If implemented, this should be tested extensively by the testing team to verify that requests only from trusted sources are accepted by the database server, and any requests from untrusted servers result in very succinct and generic error messages.
- Instead of assigning access directly to the tables, providing access through user-created views or stored procedures goes a long way in securing a database against attacks. This approach allows a user read-only or indirect access through safe, pre-designed methods, thereby reducing or eliminating data manipulation capabilities (Splaine, p.177)
- Locking down access control, and using principle of least privilege are critical to secure a database against attacks. As discussed in privilege elevation section, this can be done by creating roles in the database, and assigning users to one of the defined roles.
- Is confidential customer information protected at the database level ? Legislations such as HIPAA (1996), and the Gramm Leach Bliley Act (2001) require corporations to take considerable precautions to keep customer records private and secure. One way to accomplish this is to encrypt data in the database, using a robust cipher with sufficiently strong encryption key. Keep in mind that encryption is never a complete security solution by itself, because encrypted data can still be tampered with or deleted, it only cannot be read. Another suggested method is to deploy audit controls at the database and schema level, so that any updates to the application data or to the schemas are tracked. Most of the modern databases support audit trail features for database updates and more, like being able to audit sensitive information retrieval, to help monitor if anyone is trying to steal important data. Though it is not a preventive measure, audit trail can be a powerful and effective feature for troubleshooting after an incident, especially in environments where unique logins for each database user and administrator are implemented. If implemented, data encryption or database audit features should be meticulously tested by testing team to verify their effectiveness.

© SANS Institute 2005 Author retains full rights.

5 General testing guidelines for web applications

Security testing is a completely different animal compared to functionality or regression testing of a web application. While functionality testing involves testing the system against a definite test plan and measurable test cases, security testing requires the tester to think like a hacker, and look for any abnormal behavior from the application. Testers need to think unconventionally or "out-of-the-box" when testing web applications for security, and should perform negative testing, trying hard to break the system, by providing unexpected input etc. Testing team should also be aware of organizational and legislative security policies and standards, and make sure that the application is in compliance with those.

Here are some general guidelines for security testing of web applications

- Developers should review the code to be deployed with security and testing teams, prior to deployment. Reviews are useful to bring out any obvious omissions or mistakes in the application code. Development and testing teams are also advised to run the application code through code coverage tools, as referred and discussed in section 6.2 *Code Coverage Tools*, to ensure that all code branches of the application are tested. It goes without saying that untested parts of code are much more likely to be vulnerable than tested parts.
- Is the application in compliance with the password rules of company's security policy, for creating and changing passwords ? This can include rules regarding lengths of passwords, enforcing expiry of passwords etc.
- Verify how the application handles default permissions on files, database tables etc. Sometimes, applications may give away privileges unless explicitly marked 'No' (like in configuration files), and letting the defaults ride in these cases may lead to a front-door entry for attackers.
- Test how the application reacts to repeated failed attempts. A secure web application should be implemented with a built-in proactive alarm system, that generates real-time alerts for repeated failed attempts. 'Real-time' is important, because if the application generates e-mails as its notifications, and a hacker attempts to break into the system during a weekend, the generated e-mails may go unread until the next business day, giving the hacker sufficient time to get his job done. In this example, a smart alarm system would generate text pages, so that attempts to thwart a possible intruder can be put in place right away.
- Testers need to be meticulous, and should have conservative criteria for test case pass/fail, while testing web applications for security. For example, a test case may pass as long as it doesn't crash the application, but it may be causing other undesired effect like holding a lock on a table, which can go unseen. In this example, if the tester passes it because it is not crashing the application with a single user, the application will crash and lead to DoS attack, once it is in production, and is hit by thousands of users. Such "behind the scenes" activity may be significant, and should be tested in detail, before the test is passed.

© SANS Institute 2005 Author retains full rights.

- Every web application uses one or more off-the-shelf components like web server, or database server. Make sure that the application team is subscribed to the vendor's distribution list for security updates, receives security advisories from the vendor in a timely manner, and necessary security patches are applied as soon as available.
- Special care needs to be taken in single sign-on environments, to protect passwords, because in such environments, by cracking a single password, enterprise-wide security can be compromised, as access is not limited to one application or system. Security testing team is encouraged to take notice of this, and to exercise extra caution while testing in single sign-on environments.
- Sometimes, forgotten password procedures can leave a backdoor open for hackers, if weak identification procedures are implemented to authenticate the user. For example, a web site that asks user's Date of Birth, mother's maiden name and two security questions is much more robust than one that just asks 'What is your favorite color' to authenticate user. In this example, the latter can be easily guessed or "brute-forced" by an attacker to gain illegal entry into the web site. Testing team should perform negative testing to find out the robustness of the forgotten password procedures, and to ensure that this is not the weakest link of the web application.

6 Useful tools for web application security testing

Security testing of web applications can be a very time consuming and tedious task. A number of free and commercial tools are available in the market, to automate various aspects of security testing of web applications. This section presents some useful tools that can be handy to testers for security testing of web applications efficiently. In addition to the tools discussed in detail here, this section also refers other available tools that belong to the same testing category.

Note that pure network security tools such as port scanners, network sniffers etc, are out of scope of this paper, as they are not a part of web application testing.

6.1 Application vulnerability Management tools

6.1.1 Cenzic Hailstorm

Cenzic Hailstorm is an Application Vulnerability Management tool that aims at knitting the Information security, Development and QA teams together, so that security policies are established, and testing happens as a part of the SDLC, not as a separate or "after-the-fact" initiative. It integrates with Mercury Interactive and other QA tools, and provides the following features (as described in the white paper http://www.cenzic.com/pdfs/CenzicWpBeyondScan.pdf)

- Hailstorm Policy modeler allows the information security team to create policies, which can be shared across the organization

© SANS Institute 2005 Author retains full rights.

- Provides mechanisms for automatic testing of some of the most common application security vulnerabilities like buffer overflow, unvalidated input, script injection etc.
- Comes with a good database of security policies out of the box, that include verification of access controls, data privacy and compliance with regulations like HIPAA, Sarbanes Oxley etc.
- Can be used for testing at various levels of granularity, like application-level, web page/form level, field level etc.

Hailstorm is a multipurpose web application testing tool, and following are some of the application vulnerabilities, which can be automatically tested using Cenzic Hailstorm :

- Unvalidated parameters
- Broken access control
- Broken Account and session Management
- Cross site scripting flaws
- Buffer Overflows
- Command injection flaws
- Error handling problems
- Insecure use of cryptography
- Remote administration flaws
- Web application server misconfiguration

For details of how Hailstorm supports the above-mentioned application tests, refer to the whitepaper http://www.cenzic.com/pdfs/CenzicWpImpAsPln.pdf. Hailstorm is commercially licensed, and is targeted for business customers.

6.1.2 AppScan QA

AppScan QA is an Application vulnerability scanning software from Watchfire (originally from Sanctum, which is now acquired by Watchfire). AppScan QA is an automated web application vulnerability scanning and testing tool, that provides QA personnel with comprehensive security defect analysis, and remediation information. AppScan provides the following features

- Checks for known vulnerabilities in commercial products (like unpatched web servers, application servers etc)
- "crawls" the web site or application, visiting links as an end user would, and reports any faulty links, or links that have not responded as expected, in a high-level report like below. Details of each category can be looked at, by clicking on the explore category.

© SANS Institute 2005 Author retains full rights.

Exploring The Site	
Visited Links	27
Unvisited Links	653
Interactive Links	1
Filtered Links	88
Faulty Links	2
Potential Vulnerabilities	694

- Clicking on *Potential Vulnerabilities* provides a detailed list of vulnerabilities detected within the application like below

Name	Test Requests	Vulnerable	Highly Suspicious	Suspicious	Not Vulnerable
Third party misconfigu	1155	18	0	0	630
Known vulnerabilities	207	5	1	0	158
Forceful browsing	855	5	0	0	763
Hidden field manipulat	4	0	0	0	0
Cross site scripting	121	0	0	0	22
Stealth commanding	9	0	0	0	0
Parameter tampering	210	0	0	0	0
Backdoors and debug	30	0	0	0	6
Suspicious contents	27	0	0	0	0

- Exploration can be done for the entire application or a part of the application containing specific business processes. This feature can be very useful for application security testing, in scenarios where an application is tested by several testers, and each tester tests only a few modules
- Automatically creates test cases which the user can run, based on potential vulnerabilities identified during the explore stage.
- Lets the user group test results using various grouping methods like category, severity, link etc. Results can be drilled down to the level of an individual result (index card), where you find detailed test description, test response, and fix recommendation including helpful URLs.
- Allows the user to create a configurable report based on test results, that can be configured for the look and feel, as well as the technical details. Generated reports can be exported into PDF, Excel, HTML, RTF, text or Tiff formats.
- Allows user to create a report for compliance with various policies/regulations like HIPAA, GLBA etc, or based on a user-defined regulation file. AppScan ships with most of the recent regulations, and this data can be updated with regular AppScan updates.
- Allows users to schedule automatic, periodic scans

The information and screenshots in this section (6.1.2) are from *AppScan QA user manual*, which is bundled with the AppScan QA product.

AppScan is commercially licensed, and can be downloaded from http://www.watchfire.com/products/security/appscan-qa.aspx

© SANS Institute 2005 Author retains full rights.

Following are some of the other application vulnerability Management or scanning tools available in the market.

- WebInspect (http://www.spidynamics.com/products/security/WI/index.html)
- Reasoning (http://www.reasoning.com/)
- CodeWizard (http://www.parasoft.com/jsp/products/home.jsp?product=Wizard&itemId=62)

6.2 Code Coverage tools

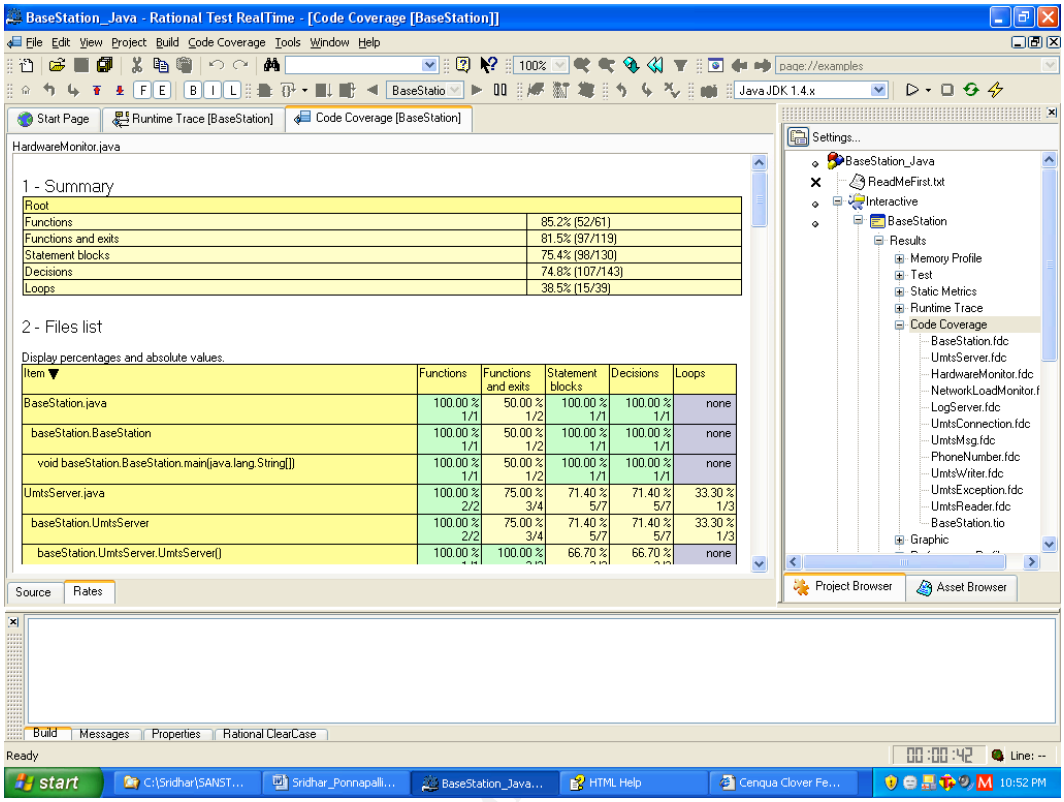
Code coverage tools primarily run through the application code and report which parts of the code have never been executed during testing of the application. Security testing should include running application code against code coverage tools, to make sure if there are any untested parts, that may be exploited by hackers. Code coverage tools can typically be categorized by the application platforms they cover (like .NET, java, C, C++), or by the level of detail of code coverage (like path coverage, branch coverage, line coverage etc). These tools, either by themselves, or working with compilers or IDE platforms, implement a technique commonly called 'instrumentation' for inserting smart probes in the code. These smart probes collect profile statistics for the code, like how many times a piece of code has been executed etc. This profile information is useful in many ways, for example for identifying which parts of code haven't been tested, or which parts of code are most used by users in production systems etc. There are many commercial and freeware code coverage tools available in the market.

6.2.1 Rational Test RealTime

Rational Test Realtime is a code coverage analysis tool that provides code coverage for Ada, C, C++ and Java. It provides the following features

- Code coverage, which identifies portions of the application code that have or have not been executed during testing. You can create a new project containing your application within Rational Test RealTime GUI, and *Build* the application. Then, run it a few times with some test data, and a report like below is produced containing the results of code coverage analysis at various levels, like functions, blocks, groups, loops etc.

© SANS Institute 2005 Author retains full rights.



After running some tests, a security tester can generate a report like this, and refine his testing to include all portions of code that haven't been tested.

- Memory profiling for C, C++ or Java code, including detecting possible memory leaks and memory usage issues with the source code
- Performance profiling for C, C++ or Java, which measures performance of each component or module, and detects bottlenecks

Note that code coverage through instrumenting may have some performance overhead on the application, because of the inserted code. So a tradeoff is required between the code coverage level and application performance, if coverage is implemented in production systems.

Here is the link for more information regarding Rational Test RealTime, and to download a trial version
http://www-306.ibm.com/software/awdtools/test/realtime/features/index.html

Following are some of the other code coverage tools available in the market.

- LDRA Testbed (http://www.ldra.co.uk/pages/testbed.htm)
  A code analysis and coverage tool suite, that can be used for C, C++, C#, Java, Visual Basic, Fortran, Pascal and other languages. Platforms supported include Windows, Linux, and Unix flavors (Solaris, HP-UX etc)

© SANS Institute 2005 Author retains full rights.

- Dynamic Code Coverage (http://www.dynamic-memory.com/coverageanalysis.php)
  A code coverage tool for Unix, this can be used for covering C/C++ code. Platforms supported are Solaris and Linux
- Clover.NET (http://www.cenqua.com/clover.net/)
  A code coverage analysis tool for .NET applications

6.3 Password auditing tools

Even though modern authentication methods like Biometric scans are increasingly finding a place in business, undoubtedly the technique that is still most widely in use to authenticate users is good old login/password. Password auditing tools are used to "crack" or guess application passwords, and/or recover lost passwords. These tools are typically used by hackers to get illegal entry into applications, and come in handy for security testers, as they wear a "hacker's hat", to check if the passwords used can be easily guessed. If passwords for an application are easily cracked (like dictionary words, easily guessable passwords like name of the organization, or can be easily brute-forced), security testers should suggest more stringent password policy, like making them longer, or enforce combinations of uppercase and lowercase letters, digits and special characters etc.

Here are some password auditing tools available in the market

- L0phtcrack (http://www.atstake.com/products/lc/) – Windows and Unix password cracking and recovery
- John The Ripper (http://www.openwall.com/john/) – Windows and Unix command line password cracker
- Cain & Abel (http://www.oxid.it/cain.html) – password cracker and recovery tool for Windows

6.4 Use Google to check if sensitive information is exposed

Today's search engines on the internet are very powerful, and sometimes this can work against your organization by exposing sensitive information to the public. As Nitesh Dhanjani advocates in his article *Google your site for security vulnerabilities* (http://www.onlamp.com/pub/a/security/2004/10/07/googling_for_vulnerabilities.html), it is a good practice to test your web site for any sensitive information left open by an application or human. This is best accomplished by doing a site specific search on Google for your site or application, searching for keywords like 'password', 'userid', 'login', 'admin' etc, and checking through the resulting pages. Here are some sample search strings.
site:www.yoursite.com password
site:www.yoursite.com admin

© SANS Institute 2005 Author retains full rights.

7 Conclusion

Securing critical internet, intranet, and extranet applications of an organization is a shared responsibility between system architects, developers, testers, quality assurance team, security team and project management teams within an organization. Security team should not be viewed as separate, standalone team for auditing applications, but should be integrated with development and testing teams, right from the concept phase of a project. Organizations should inculcate the doctrine “it is a matter of when, not if, you are attacked”, in its employees, and make sure each employee plays a role in defending the corporation against attacks. Adding robust security testing to the lifecycle of web applications demonstrates defense-in-depth, as this adds one more important layer to the defense infrastructure of the organization, in addition to network security, and secure architecture, design and coding practices.

8 References

1. Splaine Steven Testing Web Security: Assessing the Security of Web Sites and Applications Wiley Publishing Inc, 2002.

2. Whittaker A. James & Thompson H Herbert How to Break Software security Addison Wesley, 2003.

3. Scambray Joel & Shema Mike Hacking Exposed Web Applications McGraw-Hill/ Osborne, 2002.

4. McGraw Gary & Viega John “Make your software behave: Learning the basics of buffer overflows” March 2000. URL : http://www-106.ibm.com/developerworks/security/library/s-overflows/ (12 September, 2004)

5. Frykholm Niklas “Countermeasures against Buffer Overflow Attacks” November 2000. URL : http://www.rsasecurity.com/rsalabs/node.asp?id=2011 (13 September, 2004)

6. OWASP Top Ten vulnerabilities, Improper Error handling URL : http://www.owasp.org/documentation/topten/a7.html (15 September, 2004)

7. [Spett1] Spett Kevin “SQL injection. Are your web applications vulnerable?”. URL : http://www.spidynamics.com/whitepapers/WhitepaperSQLInjection.pdf (17 September, 2004)

8. [CERT1] CERT Coordination Center “Denial of Service Attacks” URL : http://www.cert.org/tech_tips/denial_of_service.html (20 September, 2004)

© SANS Institute 2005 Author retains full rights.

9. [CERT2] CERT Advisory CA-2000-02 “Malicious HTML Tags Embedded in Client Web Requests”
URL : http://www.cert.org/advisories/CA-2000-02.html (20 September, 2004)

10. [Spett2] Spett Kevin “Cross-Site Scripting. Are your web applications vulnerable?”.
URL : http://www.spidynamics.com/whitepapers/SPIcross-sitescripting.pdf
(24 September, 2004)

11. Fallin Steve & Pinzon Scott “What We Mean by “Elevation of Privileges””
URL : http://www.watchguard.com/infocenter/editorial/135144.asp
(26 September, 2004)

12. [Cenzic1] “Beyond simple vulnerability scanning” May, 2004
URL : http://www.cenzic.com/pdfs/CenzicWpBeyondScan.pdf (2 October, 2004)

13. [Cenzic2] “Imperative Web Application Assessment Plan” May, 2004
URL : http://www.cenzic.com/pdfs/CenzicWpImpAsPln.pdf (4 October, 2004)

14. Dhanjani Nitesh “Google your site for security vulnerabilities” October 2004
URL :
http://www.onlamp.com/pub/a/security/2004/10/07/googling_for_vulnerabilities.html (15 October, 2004)

15. Watchfire Corporation “AppScan QA User manual” 2004

Links to Security Testing tools

16. Software Research, Inc Home page. URL : http://www.soft.com/ (15 September, 2004)

17. KSES download page
URL : http://sourceforge.net/projects/kses (25 September, 2004)

18. SPI toolkit from SPI Dynamics.
URL : http://www.spidynamics.com/products/Comp_Audit/toolkit/index.html
(18 September, 2004)

19. Spike proxy from Immunity
URL : http://www.immunitysec.com/resources-freesoftware.shtml
(18 September, 2004)

20. Mercury LoadRunner Home page
URL : http://www.mercury.com/us/products/performance-center/loadrunner/
(21 September, 2004)

© SANS Institute 2005 Author retains full rights.

21. Empirix e-Load Home page. URL : http://www.empirix.com/Empirix/Web+Test+Monitoring/Testing+Solutions/Web+Application+Load+Testing.html (24 September, 2004)

22. Empirix e-Test suite Home page. URL: http://www.watchguard.com/infocenter/editorial/135144.asp (27 September, 2004)

23. Watchfire AppScan QA Home page URL : http://www.watchfire.com/products/security/appscan-qa.aspx (9 October, 2004)

24. SPI Dynamics WebInspect Home page URL : http://www.spidynamics.com/products/security/WI/index.html (10 October, 2004)

25. Reasoning Home page URL : http://www.reasoning.com/ (12 October, 2004)

26. Parasoft CodeWizard download page URL : http://www.parasoft.com/jsp/products/home.jsp?product=Wizard&itemId=62 (13 October, 2004)

27. L0phtCrack product homepage URL : http://www.atstake.com/products/lc/ (14 October, 2004)

28. John the Ripper Home page URL : http://www.openwall.com/john/ (14 October, 2004)

29. Cain & Abel Home page URL : http://www.oxid.it/cain.html (14 October, 2004)

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005 Author retains full rights.