White Paper

# Stateful Assessment™

## A Better Way to Identify Vulnerabilities in Cloud, Mobile and Web Applications

## Table of Contents

Executive Summary ....... 3
Background ....... 3
Limitations of First Generation Application Testing Tools ....... 3
Signature-Based Technology Flaws – False Negatives ....... 4
Signature-Based Technology Flaws – False Positives ....... 4
The Stateful Assessment Approach ....... 5
About Cenzic ....... 10

## Executive Summary

First generation application security vulnerability scanners employed an approach based on the use of signatures (matching of regular expressions) to detect vulnerabilities. This paper will explore the limitations of signatures and introduce examples of a more effective approach – Stateful Assessment®. Stateful Assessment is based on a process of generating Cloud, Mobile and Web application transactions and evaluating the response of the browser over the course of the entire transaction (for native mobile apps, a proxy is used to emulate a browser). The advantage of this approach is a dramatic increase in the number of vulnerabilities found, a decrease in false vulnerabilities (false positives), and validation inherent in the process.

## Background

Detecting vulnerabilities in Cloud, Mobile and Web applications before hackers can find and exploit the flaws is a critical element of application vulnerability management. These applications are susceptible to a wide range of critical flaws that, when exposed and exploited by an attacker, can result in serious damage both to a company (in terms of direct financial loss both from the attack and in recovering from its aftermath) as well as to the users of the application who may become victims of fraud and identity theft.

In order to protect mission-critical applications from exploitation, companies often rely on security technologies to aid in the process of finding security flaws. Application security vulnerability scanners have become the most prevalent method of identifying vulnerabilities.

## Limitations of First Generation Application Testing Tools

The major disadvantages of first generation tools stem from a lack of intelligence within the assessment process. In addition to lack of finds and false positives, the final report often contains a low signal-to-noise ratio in terms of actionable information. Improvements in the underlying intelligence of the scanning and vulnerability recognition process will yield benefits in several areas:

- Less time and manual effort will be required to post-process the results of the assessment
- The results that are obtained will be more actionable
- The response time will be reduced

The limitations of first generation tools have the common thread of a "signature dependent" scanning mechanism in which static, or unchanging, signatures are the vehicles of vulnerability detection. Signature-dependence indicates a way of detecting a vulnerability based on a codified pattern.

As a pattern, a "signature" of a vulnerability is a string of characters returned by an application, usually in response to a probe of one kind or another that is used to infer the presence of a security vulnerability. Detection of the vulnerability is then a function of pattern matching: comparing the application's response to a list of possible signature values, and then assigning vulnerability based on the result of that comparison. Presenting an example signature will help make the point more clear.

Consider the following database error message:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
```

One could use this message as the basis of a signature. Although this is a simple example, consider a few of the challenges inherent with the task. First, as an indicator of vulnerability, the full string would be highly specific, i.e., it is a particular type of error message `'80040e14'`. Using the entire string would therefore only match the signature when error message `'80040e14'` was generated. If the application returned a different error code, say, "`80040e14`", the string would not match and no vulnerability would be inferred. Thus, the chosen signature, due to its specificity, would match only in very rare conditions, namely, when probes by the scanner generated the exact string and error message above.

## Signature-Based Technology Flaws: False Negatives

One of the common disadvantages of first generation application testing tools is the problem of false negatives. Often, applications return information in response to probes that simply lies outside of the scope of the signature base, resulting in actual vulnerabilities that are not detected. This problem is addressed, although not particularly well, by adding more signatures. The problem space is simply too large to anticipate and codify ahead of time the full range of Cloud, Mobile and Web application behavior in the real world. A signature-set, no matter how large, will always come up short and when it does, false negatives will occur. The only real answer to this problem, architecturally speaking, when signatures are optimal for detecting a particular type of security flaw, is to implement an extensible rule set that the user can quickly and easily modify. First generation tools are extensible only in principle, requiring the user to learn a custom programming language similar to C or C++, and then compile their new security test into a module or add-on, a process as cumbersome as it is impractical.

Content aside, our example signature is a signature of an error message, and thus its appearance is subject to the application (and server configuration), and can be suppressed by graceful error handling. A large part of the problem of false negatives results from the fact that the signature itself depends on conditions that are only observable under the same configuration and patch-level conditions that produced the original content from which the signature was taken. Other vulnerabilities, due to their place within application internal logic, or due to the presence of intermediate computer systems or other applications, give no immediate or observable response to probing activity, thus enlarging the scope and frequency of the problem of false negatives.

Content-wise, the most common approach to reducing false negatives by first generation tool vendors and open-source technologies has been to make signatures more general, and thereby catch more cases in which the signature is matched. Operating on the basis of a fairly small set of generic signatures for each vulnerability type has become the de facto industry standard for detecting vulnerabilities like Cross-Site Scripting and the varieties of SQL injection. Once again, this does not eliminate false negatives, as small, generic, signature sets are still in principle signatures of this or that under these conditions, but more generic signatures do broaden the conditions under which a successful pattern match can be made.

As a more generalized signature, obtained in this case by shortening the original error message string, the chances of a successful pattern match are improved, as far more examples of “Microsoft OLE DB provider” are likely to appear than the full string from our first example. However, a successful pattern match in no way guarantees a successful inference of vulnerability, chiefly, because of the tendency of generic signatures to be triggered by things other than the vulnerability itself. In practice this can happen with the response received from the application under test contains the signature but the string, in virtue of its generality, is part of existing HTML or application response content. The sentences, “The Microsoft OLE DB Provider is a piece of junk!” or “The Microsoft OLE DB Provider is a work of art!” would both trigger a positive match from our shortened signature.

## Signature-Based Technology Flaws: False Positives

Typically, generic signatures introduce the second major disadvantage of the signature based technologies - the problem of false positives. A probe elicits a response from an application that that matches the signature, but the signature match is simply erroneous evidence of a vulnerability due to the presence of other conditions. One notorious example of this type of false positive occurs when a scanner uses the string “200 OK” to infer the presence of a file, the signature being derived from the HTTP status code indicating success or resource found. When custom 404 error handlers are in place, “200 OK” is returned for non-existent files, so the tool erroneously reports that every insecure file in its vulnerability database resides on the target.

In other words, the conditions in which the signature will match are often reproduced by a particular application or server configuration, and as a result, the reliability of generic signatures is questionable. Mis-configured applications and application errors are frequent culprits in false positives, and the more general the signature, the greater the likelihood that the signature will be triggered by something other than the specific conditions of vulnerability.

Another type of false positive occurs because too much is expected of the signature, or more specifically, the signature is neither necessary nor sufficient for reliably inferring vulnerability.

One example of this type of signature inadequacy comes in the detection of SQL vulnerabilities that allow an attacker to run and execute arbitrary SQL queries. Because successful queries often look different across any two applications, database errors are frequently used to infer the ability to run SQL commands. Errors, being fairly standard, are often used for signatures. However, a generic database error can result by modifying an application input, yet no ability to run unauthorized SQL commands exists. Even the correct pattern matching of the actual database error that the signature is a signature of, fails to give strong evidence of any deeper vulnerability. A large number of potential vulnerabilities result from security scans with signature-based technologies. The signature, as matched, isn't enough to be a strong indicator of vulnerability, and more investigation is required. In these cases, a better or more reliable signature doesn't help, because pattern-matching of the particular error or response information doesn't tell us much about vulnerability itself. In these cases – and they are quite common – the limits of the signature-based approach is clearly revealed, because the dynamic behavior of successful exploitation defies clear-case signatures. One finds this type of limitation exposed most clearly when testing for a wide range of application vulnerabilities, from "deep" SQL disclosure vulnerabilities, application logic vulnerabilities, and application session management security. These complex vulnerability types don't lend themselves readily to signature crafting.

## The Stateful Assessment Approach

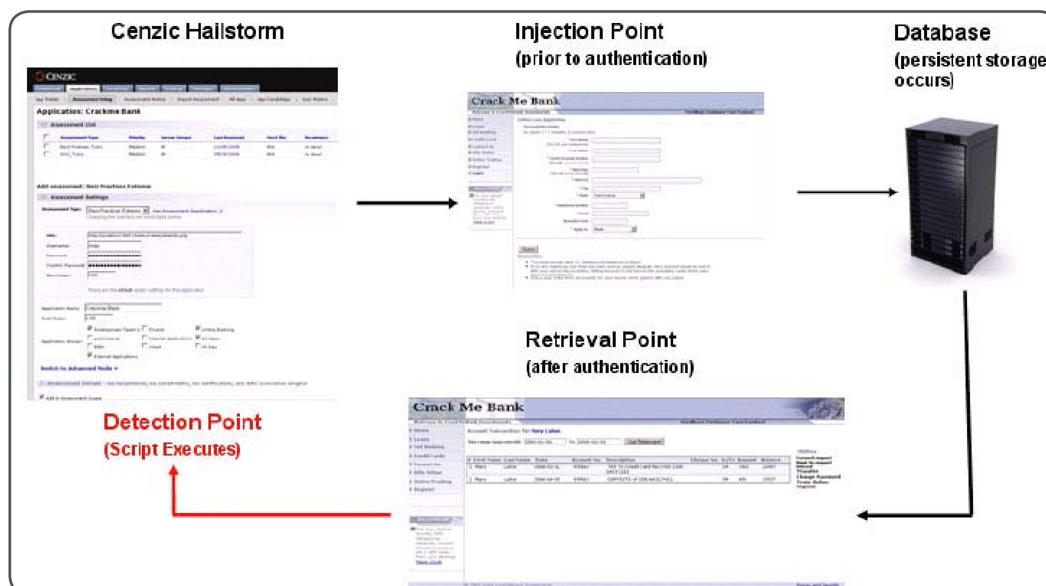
When confronted by some of the more complex problems in application security, the natural reaction is to believe that certain vulnerability types are best detected by problem solvers. Cenzic Hailstorm's Stateful Assessment technology is a breakthrough based on utilizing intelligence in the assessment process. This intelligence is obtained by better maintaining state between the application being assessed and the localized point of assessment.

Maintaining state with the application during the testing process gives the assessment engine additional information that is ordinarily not available within the immediate responses of the application to a particular probe or query. Rather than deploying an agent on the system under testing, Stateful Assessment leverages the interdependent relationship between the browser and the application. The browser thus functions as a lens through which to view the application's state through the synchronization that occurs between the browser and the application. Additionally, many application vulnerability conditions can be readily detected by browser-level events.

Stateful Assessment improves the assessment process in several key aspects. The maintenance of state with the application frees the WASVS engine from a dependency on signatures. Additionally, the WASVS engine is no longer limited by the immediacy of the application's response during the testing process (non-localized detection). Lastly, improvement results from more advanced heuristics, greater accuracy, reduction of false positives, and the enhanced validation of results.

One example of the signature independence obtained via Stateful Assessment is evident in Hailstorm's security testing procedure for Cross-Site Scripting (XSS).

Small scripts are injected into the application during the security testing process, each script having its own unique "watermark" coded into the script at run-time. The diagram below shows an example flow of the testing process, with a script being supplied to the application as input at a time prior to authentication.




In the example above, the injection of the script results in that script being stored persistently by the application's database. Later in the testing process, following an authentication sequence, a page is processed that results in the retrieval of the stored script from the database. As this information reaches the Hailstorm browser the script executes and this execution is detected via the watermark as a browser event. Monitoring java script events within the browser memory allows for the detection of previously injected scripts. As a result, a more complete picture of the vulnerability is attained than is possible with other technologies. The vulnerability can not only be detected, but the injection point or what we can identify as the localized vulnerability can be traced all the way through to the point of execution, that is, the place within the application where the malicious content is finally served up to the user.

The non-reliance on signatures is evident in the watermarking of the scripts, i.e., detection is not a function of pattern matching. The new mechanism, the "watermark", far less likely than an ordinary signature to result in a false positive, as it is generated at run time and is random. Moreover, detection is tied to the script executing in the browser, hence detecting the vulnerability involves an intrinsic level of verification that is not present during pattern matching: the execution of the script verifies that the script syntax was preserved intact by whatever encoding, storage, and retrieval functions performed by the application. In other words, detection of the script executing in the browser is evidence of the script executing in the browser, hence, the cross-site scripting vulnerability has simultaneously been detected and verified. With a signature, the signature would be evidence of a particular string of characters contained within the HTML output of the application, but the exploitability of the vulnerability would still need to be verified or validated separately.

The example detection process presents functionality going beyond the first generation tools in another respect. Most existing technologies are blind to application responses that do not immediately follow the probing activity: a particular test is performed, the result is obtained, and the process continues. Not surprisingly, existing technologies are very poor at finding the most serious kind of Cross-Site Scripting vulnerability, namely, cases involving the persistent storage and later execution of a malicious script. The stimulus-response nature of current technologies is evident when an application gives no immediate response relevant to the signature set of a particular scanner. Since an application's response that directly follows the test is benign, that is, nothing bad appears to have happened, the security test fails to detect a vulnerability. False negatives are introduced in virtue of the fact that not every vulnerability responds immediately to stimulus, nor do deeply embedded logic vulnerabilities have a well-defined signature. Vulnerabilities do not always behave as expected.

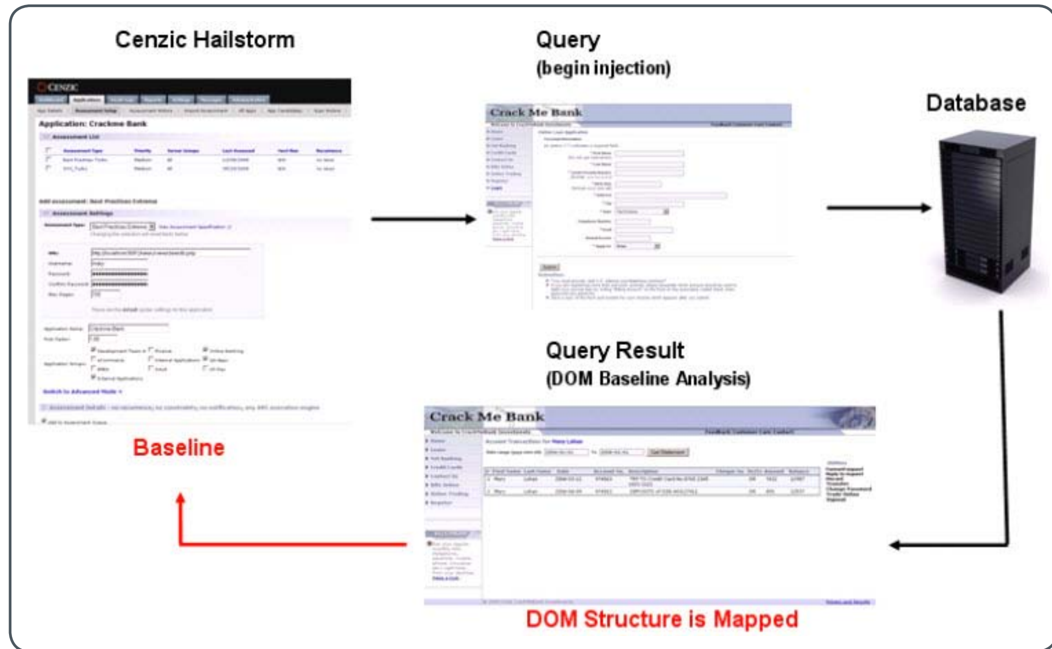
Below are reporting snapshots for each of the types of Cross-Site Scripting as distinguished by Hailstorm's Stateful Assessment testing mechanism for this vulnerability type. Below is an example of an XSS vulnerability detected from within the deep structure of an application.

When reflected XSS vulnerabilities are discovered Hailstorm indicates this difference in the message text provided to the user upon detection (shown below).

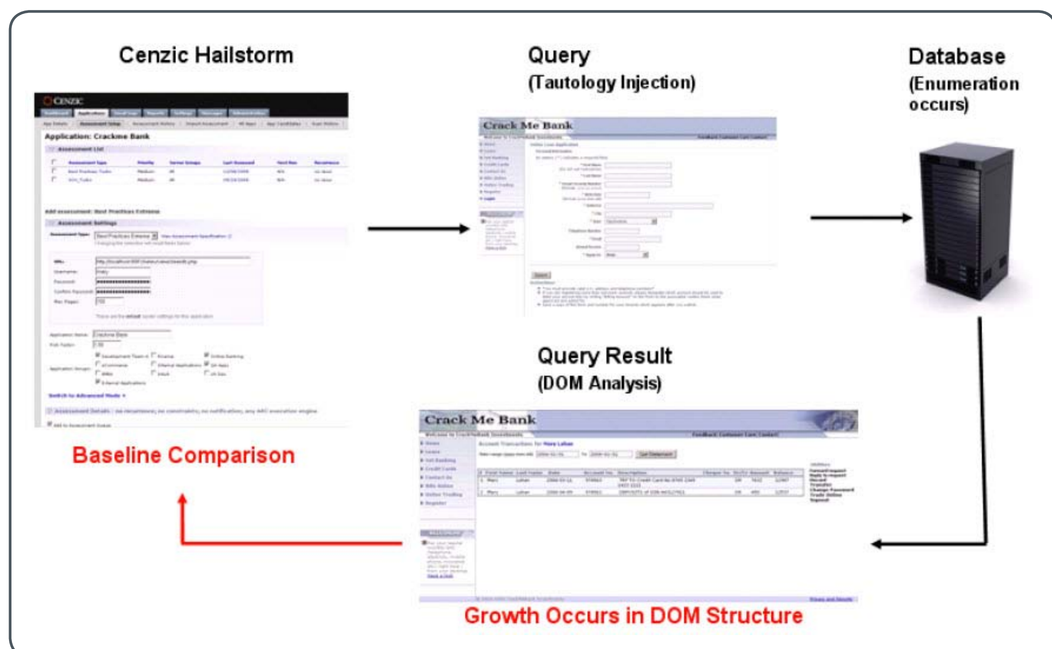
Distinguishing these types of vulnerabilities automatically is not an academic exercise, but is required for accurate risk analysis. The ability to determine the position of a script injection vulnerability, both in terms of its origin and its point of impact where the invocation of the script occurs, makes it possible, for the first time, to trace out the complete map of a XSS vulnerability, including its potential for cross-user impact.

Another example of Stateful Assessment technology is the testing process for SQL Disclosure, a SmartAttack type that explicitly tests for the ability to run unauthorized SQL queries on an application's back-end database. This type of injection flaw is particularly unsuited for detection by signature, as successful SQL queries will not be accompanied by predictable application responses. SQL Error messages generated via parameter tampering fail to accurately distinguish the vulnerability conditions which allow for full interactive access to the database server from those in which the input simply confuses the SQL parsing engine. At times, critical characters are stripped out, or escaped, such that it is garbled input, rather than a coherent string of characters, that generates the error. Additionally, triggering an error message is still merely that, an error, and the inference of deeper SQL vulnerabilities based on surface errors is frequently incorrect, contributing to more noise in the audit results.

Within Cenzic Hailstorm, SQL Error Solicitation has its own SmartAttack, separate from the SQL Disclosure SmartAttack, the latter targeting vulnerabilities that allow for interactive SQL manipulation and data mining.  The Stateful Assessment process for detecting SQL Disclosure vulnerabilities follows two phases, phase one shown below.




Once again, the browser plays a key role within the Stateful Assessment mechanism, this time; it's the DOM – the browser's Domain Object Model – that is used in detection. A DOM-based baseline is generated from a benign or ordinary query, the result of which populates the DOM. The resulting structure is used to generate a baseline against which successive non-benign queries are compared, shown below.



In phase two, tautologies are injected for the purpose of modifying the syntax of the application's underlying SQL query, causing the query to evaluate as true for every row in a database table. When successful, every record in a database table will be returned in response to the modified SQL expression. The volume of response data will be reflected by the systematic growth in the DOM structure, facilitating a comparison against the baseline and allowing for the heuristic detection of SQL disclosure vulnerabilities. The figure below shows a reporting snapshot of the successful detection of a SQL database vulnerability using this method.



Detection, in this case, occurs based on the presence of 14 additional table rows in the DOM that were not present in the baseline. This type of detection is volumetric in relation to the DOM, thus avoiding the use of signatures based on indirect evidence. A similar mechanism exists for the detection of blind SQL vulnerabilities. Although this topic will not be discussed in detail, the Stateful Assessment mechanism involves comparing the effects of two types of SQL manipulation strings, one type always true, the other type, always false. Differences in the DOM related effects of these two injections yields a basis for detecting modifications of the application's internal SQL logic.

Stateful Assessment mechanisms exist for a wide range of SmartAttacks covering other vulnerabilities not discussed, including HTTP Response Splitting, Blind SQL Injection, Cross-Frame Scripting, and others. Because each vulnerability type differs, the Stateful mechanisms used in the discovery and detection process differ in each case, such that each SmartAttack has unique, localized, Stateful Assessment mechanism relevant to the vulnerability.

## About Cenzic

Cenzic provides an application security intelligence platform to continuously assess Cloud, Mobile and Web vulnerabilities. This helps brands of all sizes protect their reputation and manage security risk in the face of malicious attacks. Today, Cenzic secures more than half a million online applications and trillions of dollars of commerce for Fortune 1000 companies, all major security companies, government agencies, universities and SMBs.

## Cenzic Products

| | |
|---|---|
| Cenzic Enterprise | Cenzic Enterprise is a software solution that assesses the security of Cloud and Web applications and supports security risk management throughout the software development lifecycle. Cenzic Enterprise provides a company-wide view of security vulnerabilities and status to executives as well as customized views for other users from a Web-based dashboard. |
| Cenzic Desktop | Cenzic Desktop is a single-user version of Cenzic Enterprise. It is designed for the power user that wants to run their security assessments on Cloud and Web applications from a single system. |
| Cenzic Managed Cloud | With Cenzic Managed Cloud, Cenzic's security experts remotely perform full vulnerability testing on Cloud, Mobile and Web applications. From a Web-based dashboard, users can request assessments, view results, run reports, analyze trends and re-test applications to verify remediation efforts. |
| Cenzic Cloud | Cenzic Cloud allows users to test their own Cloud and Web applications for basic attacks and receive actionable results all within their own Web portal – no security experts needed. |
| Cenzic Hybrid – Software + Cloud | Cenzic Hybrid provides access to both Cenzic Enterprise software and Cenzic Managed Cloud services. Vulnerability testing can be done either by using the software on premise or by leveraging Cenzic's expert security services team. |
| Cenzic Mobile | Cenzic Mobile service is delivered as a managed service. Cenzic's security experts remotely perform full vulnerability testing on mobile applications. All testing is managed from a Web-based dashboard where users can request assessments, view results, run reports, analyze trends and re-test applications to verify remediation efforts. |
| Cenzic Services | Cenzic services and training help those responsible for Cloud, Mobile and Web application security to implement best practices and procedures to protect data from hacker attacks. |

**Cenzic, Inc. | 1-866-4-CENZIC (1-866-423-6942) | request@cenzic.com | www.cenzic.com**

© 2012 Cenzic, Inc. All rights reserved. Cenzic, Hailstorm, Stateful Assessment, HARM, and SmartAttack are registered trademarks of Cenzic, Inc.