PrimeFactors™

APPLIED DATA PROTECTION

WHITE PAPER

Best Practices for Anonymizing Data in Development and QA Testbeds

August 2022

Development/QA Has Changed

Practices for IT development and quality assurance have foundations that extend much further back than the original invention of information technology. The beginning lies with Walter Shewhart of Bell Labs who, in the 1930s, introduced the concept of plan-do-study-act (PDSA) to development processes. The method was innately cyclical, iteratively seeking to identify impediments, set plans to remove them, and then measure the impact of removal on outcomes. W. Edwards Deming greatly popularized these statistical control techniques through his work in post-war Japan, where they contributed significantly to revitalizing a shattered industrial economy.

All of these antecedents informed Glenford Myers, whose work at IBM strove to formally segregate IT testing (for the purposes of identifying code deficiencies) from development engineers debugging. His approach, laid out in The Art of Software Testing in 1979, described an orderly progression of steps for methodically progressing software from initial development to final deliverable. Its basis on PDSA ensured natural points at which critical risk management actions, such as the steps for the protection of sensitive, copied data, could be documented and then performed.

At heart, all of this foundation innately assumes that top-down controls ensure an orderly sequential process. That was effective for decades, particularly during the era of centralized processing managed by few users (the legacy mainframe “glass house” data center). However, with the advent of personal computers and decentralized processing, different development models emerged that addressed the desire for reduced-time-to-market. These models use techniques that depended less on top-down command and control of development processes and more on constant communication and close coordination between development team members. These new techniques (referred to as ‘Agile’) have proven popular and, to a degree, effective. Their combination with new technologies (relational database systems, fourth generation development languages, multi-tier architectures, etc.) has resulted in development processes and QA environments that are managed by groups whose focus is more on short sprints to achieve set goals than on careful execution of a process framework.

The consequence of this evolution is that the original approach for IT development and quality assurance depends upon persistent development and QA roles, adherence to methodical workflows, and checklist documentation.

These postures align poorly with new, popular development & delivery techniques. Moreover, checkpoints that might have easily been observed under a command and control structure, such as steps aimed at protecting the privacy of production data moved to a development or QA testbed, may be less effective when work is distributed and shared, and priorities align with quick execution rather than orderly execution of a process. As one writer noted, “There is always a balance to be struck between scientific management’s goal of formalizing the details of a process (which increases efficiency within the existing technological context) and the risk of fossilizing one moment’s technological state into cultural inertia that stifles disruptive innovation (that is, preventing the next technological context from developing).” When the old way of doing development and QA does not fit the new market reality, there is a tendency for participants to throw out all of the old, neglecting those portions of it that ensured necessary protections.

Development/QA Risk Management

Human nature is such that workers tend to focus most acutely on the actions most closely aligned with their base and discretionary compensation. Developers think first about creating new components of functional code, integrators think first about efficiently integrating components, and QA testers think first on executing test plans, and therein lies the source of one important and potentially very expensive risk in almost all contemporary development projects. While there are dedicated engineers and testers involved, few projects are big enough to justify a dedicated resource to manage the data security of the testbed. While the data protection aspects of the software developed may be directly documented in the requirements for projects, the data protection requirement for how the new software is developed are seldom expressly recorded and, even when they are, these requirements tend to not be the primary focus of the development/QA team members.

In the rush to meet Agile end-of-sprint or other deadlines, project members tend to focus on their specifically assigned deliverables in their field of competence. As competition and job pressures have increased, it has become all too common that appropriate steps to anonymize copies of production data are sometimes overlooked. When a critical milestone is near and a code error corrupts the test database, the impulse is to quickly resolve the code deficiency, refresh the test database, and move on. If the sensitive data in the testbed was anonymized after it was loaded to the development or QA database, the refresh can easily leave sensitive data exposed and subject to breach.

Any successful strategy for ensuring the security of sensitive data in test or quality assurance testbeds must fit naturally into the work processes of the project team to assure that they are applied transparently even during times of deadline stress.

Testbed Risk Mitigation Approaches

Fortunately, there are three useful strategies that can be applied to copies of production data used in development and quality assurance environments: masking, tokenization, or encryption.

Data Masking

Data masking applies a permissions system that dictates what views of data are presented to engineers or QA testers. These permissions are integrated into the API the developers use to manage data in the data store (RDBMS, flat file, etc.). Assume a scenario where a project team is enhancing a customer call center application for a Visa® or MasterCard® issuing bank and the testbed includes records that contain live credit card numbers. A record imported from production into this test bed that contains a Mr. Jones' credit card number (in bold) might look something like:

53455**4660331234568338**JonesJohnC123 Any Street…

Since PCI DSS regulates the degree of security that must be applied to that data element, full card numbers should never be exposed in their entirety to any member of the development or QA teams. By use of a permissions-based masking system, the view of that record when called by any of the development or QA teams' applications should be masked as below:

- **53455466033*****8338**JonesJohnC123 Any Street…
- **5345546603300000008338**JonesJohnC123 Any Street…

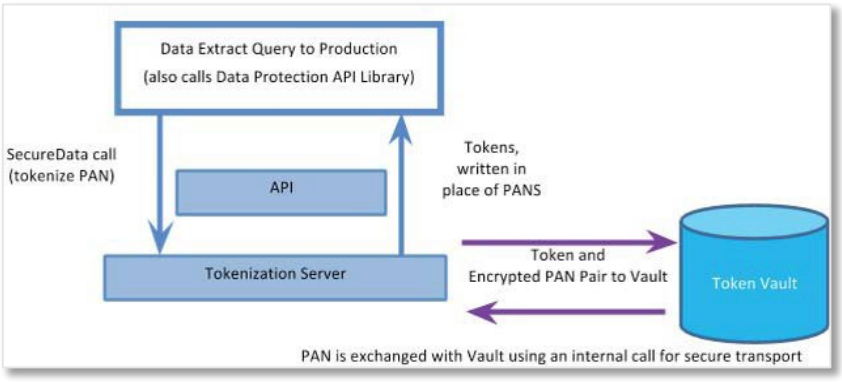
The six-digit Bank Identification Number (BIN) at the beginning of the credit card number and the last four digits remain exposed, but the middle six digits are masked, using asterisks, zeroes, or any other selected value, as may be appropriate in a given situation. Well-designed masking applications provide the further flexibility to provide different masked views of the data using role-based access controls that tailor the mask based on the role of the user or application accessing the data. For example:

- **5345546603312345678338**JonesJohnC123 Any Street…
  Allowed only for the trusted data security professional.
- **5345500000000000008338**JonesJohnC123 Any Street…
  Offered to the developer working on call center screens where the last four digit of the card number are exposed to use during cardholder identification – length and character type remain the same, so screen edits do not have to change.
- **53455466033*********JonesJohnC123 Any Street…
  Assigned to a batch code engineer for a program that uses the BIN number for sorting and routing of transactions.

Data masking has the advantage that underlying data remains exactly the same as it appears in the production system, so development systems or QA testbeds can simulate the production environment with very high fidelity. It is worth noting, however, that data masking by itself does not address protecting the data in its original storage state in the underlying datastore, potentially leaving it subject to accidental breach during backup or archival processing, or fraudulent use by datastore administrators

Tokenization

Tokenization replaces sensitive values with similarly constructed surrogate values, known as tokens, at the time of import. The unique relationship between the original sensitive value and the surrogate token value is often maintained in a highly secure token vault that project members are not granted direct access to, except by means that ensure all activities can be logged for later review or audit:



The sensitive data is never loaded to the development or quality assurance testbed and can never be exposed to the project team members in these systems. As part of the SQL import, an automated action to map an assigned surrogate token to replace the original value takes place:

Original Value	Surrogate Token
0123456789123456	8138858025938296
3456920395678345	1403022676979738
6489506573054928	4142420079004334
4758694857663956	2080136294688795

The mapping of the original value to its surrogate token remains available in the token vault for those rare instances when the integrity of the token substitution needs to be validated against the production instance.

Practical implementation options generally include token generation method (encrypted value, random number generation), generation timing (pre-generated, generated on-demand), character set, length, and check value generation. Since the token completely replaces the original sensitive value in the testbed, it can be critical that the surrogate reflect the original in many of these latter details, in order to complete with data type edits in the database or other datastore, field edits in online screens, data integrity edits in batch routines, etc.

For environments requiring extremely high levels of protection applied to the token vault, organizations often select tokenization options that protect the token vault's decryption keys using a hardware security module (HSM), such as those available from nCipher Security.

For more information, please visit www.primefactors.com.

WP-ER-ANONDATA-20220816