

SaaS SECURITY AND PRIVACY





TABLE OF CONTENTS

> 1.0 Introduction	1
> 2.0 Identity Management	2
2.1 Overview	2
2.2 Simple Internal Authentication	3
2.3 Simple Federated Authentication (Single Sign On)	3
2.4 SSO Without Federation	4
2.5 Authorization	5
> 3.0 Data Storage	5
3.1 Encryption by Customer	6
3.2 Database Instances	7
> 4.0 Data Transmission	8
> 5.0 Physical Security	9
> 6.0 Additional Security Considerations	9
6.1 Audit Trail	9
6.2 Disclosure of Provider's Proprietary Information	9
6.3 AJAX	10
6.4 Multifactor Authentication	10
6.5 Patches	10
6.6 Generation of Keys	10
> 7.0 Summary	11
> 8.0 References	11
> 9.0 Glossary of Terms	12




1.0 INTRODUCTION

This paper is one of a series of papers that explore and discuss the technical architectural components of the Software-as-a-Service (SaaS) model.

SaaS shares the distinction of being both a business model and an application delivery model. SaaS enables customers to utilize an application on a pay-as-you-go basis and eliminates the need to install and run the application on the customer's own hardware. Customers generally access the application via a Web browser or thin client over the Internet. SaaS is most often subscription-based and all ongoing support, maintenance, and upgrades are provided by the software vendor as part of the service. Application customization capabilities, if available at all, are generally provided to all customers in a consistent manner. From the perspective of the software vendor, the SaaS model provides stronger protection of its intellectual property, operational control of the environment running the software, and generally a repeatable revenue stream from the service subscription fees. Software vendors have varying capabilities and applications can come in varying flavors but SaaS applications most typically support many unique customers using a single instance of that application — also known as multi-tenancy.

SaaS requires more care around security than any of other available delivery models. The current best security practices associated with application development involve a layered approach. Regardless of the software delivery model, security cannot be implemented at a single “make or break” point. Instead, security must be layered into the network, the servers, the code and the database. Security comes in the form of both active prevention and, more recently, intrusion detection. This paper addresses the security concerns unique to the SaaS business model and assumes a working knowledge of general best practices for application development.

SaaS solutions are chiefly differentiated from the Application Service Provider (ASP) model or “in-house” applications by their unique quality of multi-tenancy. The ASP model forced business executives to confront their fear of putting mission-critical information on third-party servers but the SaaS model compounds that fear by comingling that data in a multi-tenant environment. For a SaaS provider to succeed they must engender confidence in their system by demonstrating superior security architecture. (The subject of multi-tenancy is discussed in more detail in a separate paper on Application Tenancy.)



Security concerns for the application delivery environment in a SaaS model share many of the same concerns as other application delivery models. The following should be considered:

- > Single function servers
- > Firewalls zones (DMZs), and Virtual LAN segments (VLANs) as appropriate
- > IDS — Intrusion detection system. A network and/or host based monitor which looks for suspicious activity against patterns of known suspicious behavior
- > IPS — Intrusion protection system. An active version of IDS which interrupts communication streams when suspicious activity is detected
- > Extensive logging of all activities from routers, firewalls, IDS, IPS, databases, Web code and application layer code to an logging server which has independent credentials from the production hardware to maintain log reliability
- > Updated antivirus on every server
- > Strong passwords
- > Each user must have a distinct login. There can be no sharing or pooling of logins
- > Services do not run under the administrator account

Where we really start to see a difference in the SaaS model in terms of security practices is related to identity management, data storage and data transmission.

2.0 IDENTITY MANAGEMENT

2.1 Overview

Given that modern SaaS architectures usually come in the form of a Web-based application, we can make several assumptions. First, communication between the user and the service provider leverages SSL or TLS encryption. Second, that communication is coming over the public Internet. Third, many (but not all) customers will be looking for a vendor capable of providing a solution built using SOA principles, as these customers want a service capable of integrating into their enterprise environment using industry standard identity management solutions. Others will want the SaaS provider to handle all security from top to bottom. This dichotomy requires we produce a SaaS solution which is capable of handling both the federated and the internal authentication model and then funnel them into a single authorization model. SaaS effectively needs to build a federated identity model from number or integration points as well as create an internal repository for customers unable to integrate their identity management product into the platform. Customization of the authorization model needs to be avoided where possible. We don't know what authentication models may present themselves in the future and generally the authorization component is interwoven throughout the core code.

Changing the authorization module requires significant effort and testing while increasing risk and should be avoided where possible. It is highly preferred that the SaaS provider not provide a custom identity management interface for each customer. It is better to offer a well tested, industry standard interface and require that customers conform to that interface. That may require additional development effort from the customer but will dramatically decrease the ongoing maintenance effort for the overall platform.

2.2 Simple Internal Authentication

The simplest method of authentication is one in which the SaaS provider maintains a user database independent of any customer's identity management.



Figure 1

Figure 1 provides a general example of a user hitting a Web site where their user right and permissions are contained in a database within that same SaaS stack. This methodology is commonly initially developed within a SaaS architecture due to the simplicity of the implementation. However, it is important to make a distinction between authentication and authorization so that incoming users can be funneled through a unified authorization module.

2.3 Simple Federated Authentication (Single Sign On)

A second option is a model in which users are able to authenticate directly from their authorizing system. Generally, this means the user is logged into an enterprise domain (i.e., Active Directory) and wants access to the SaaS provider without the need for a second login. This allows the customer to enforce all necessary security policies around identity management with the additional benefit that the resources and risk associated with identity management are now being transferred to the customer. Single Sign On (SSO) can be achieved through token (Kerberos, etc.) or assertions (SAML). For this paper, Security Assertion Markup Language (SAML)¹ is utilized due to its well established place within the Internet community. Where customers often use LDAP within their organization, it is SAML that provides the mechanism for that identity information to be safely transmitted to the SaaS provider. As the SaaS provider matures, one project which is worth watching is the Shibboleth project which will be the next generation of the OASIS SAML specification².

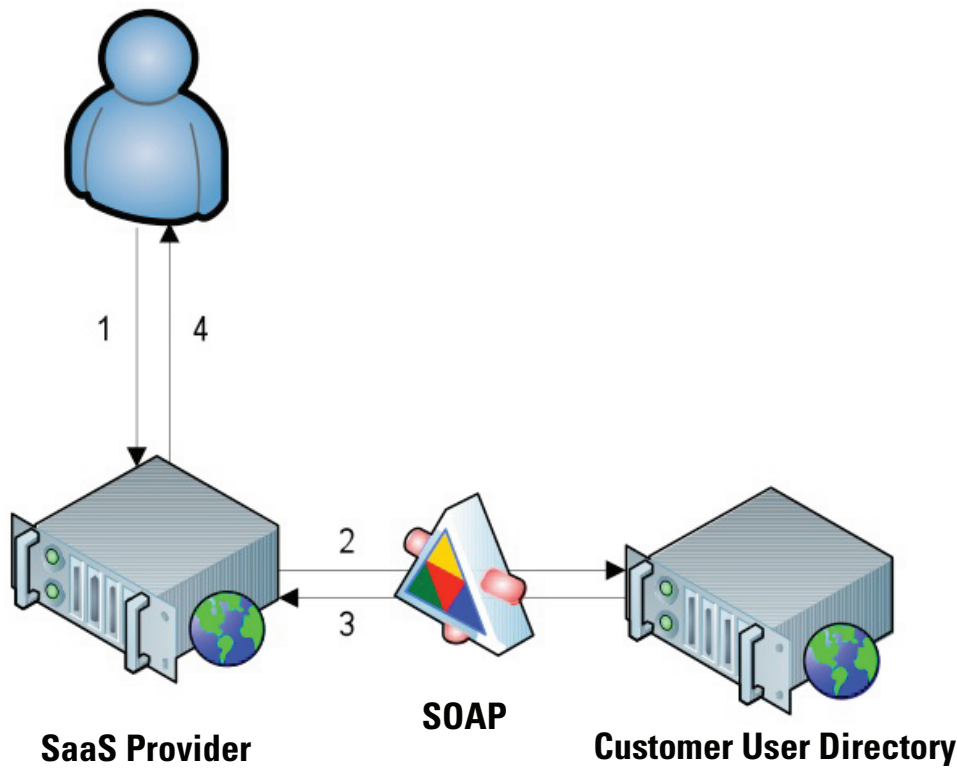



Figure 2

Figure 2 illustrates an extremely simple version of an SSO:

1. The user attempts to access the SaaS provider and will need to do so with some form of identifying information. For example, in the event the SaaS platform is Web based this may be encrypted data in the URL or a cookie.
2. That information will be authenticated against the customer's user directory via a direct Web service call.
3. The customer's user directory will then reply back with an assertion containing authorization and authentication information.
4. The resulting request is either fulfilled or denied based on the authentication and authorization of the assertion.

2.4 SSO Without Federation

A third option to consider is pre-establishing trust. If the SaaS provider delivers a public encryption key to the customer's authentication server then the user can make their initial connection to the SaaS provider with the assertion pre-encrypted as posted data. This option represents a compromise. It produces a lower level of security than a pure SAML model because it requires keys be exchanged on a regular basis and removes the aspect



of server-to-server communication to produce the token. It does, however, drastically reduce the integration complexities between the SaaS provider and the customer. A complete SAML implementation requires the customer produce a Web service on their site for the SaaS provider to retrieve information from. For many customers, this step may be more effort than they are willing to expend.

2.5 Authorization

Once a user is successfully recognized by the SaaS provider's system, it is then necessary to allow the user access to only the data and functions they are permitted to access. This is referred to as authorization. Given that a SaaS platform will mature over time and may well change or add to its identification methods, it is necessary to make sure the authorization module is separate from the authentication module(s). The authorization module should log each and every attempt at performing an action regardless of the success of that attempt. The exact mechanism of authorization is entirely dependent upon the SaaS software architecture. Remember, it is best to validate the requestor's permissions with every single request. This will help to cut down on cross-scripting attacks etc.

3.0 DATA STORAGE

In an ASP model, each customer potentially has unique hardware, keeping data segregated at all times. In a SaaS model, heterogeneous data may reside within a single, co-mingled instance of a database. To some extent, the current regulatory environment addresses heterogeneous data within a SaaS environment. PCI and HIPAA, for example, have distinct requirements around segregation of data. Both address segregation of data or require compensating controls for heterogeneous data. A compensating control is a mechanism built into the architecture designed to solve the underlying intention of a regulatory requirement even if it does not fit into a "cookie cutter" design. However, how data is stored is a key concern of SaaS customers, especially with respect to financial transactions. Multi-tenant database architecture means comingling of sensitive data will occur. This is an area that therefore needs to be proactively addressed by the SaaS provider. It is necessary to demonstrate a segregation of customer data without building distinct infrastructure for each customer both to satisfy the regulatory environment and also to provide customers with peace of mind.

Assuming the industry best practice of utilizing a three-tier architecture, it is possible to keep both the top two tiers completely stateless (in fact, you may find you are able to keep these tiers disk-less). By not keeping sessions in memory or writing to disk, these two tiers dramatically reduce their security risk. Session state then can reside between the database tier and encrypted cookies or URL information on the browser side. This methodology is language and platform independent. While this introduces more overhead

on the database tier it does simplify the construction of load balanced server farms as there is no need to keep stickiness to any given server. For Java this means no bean persistence and for .NET, no session state. It may be necessary to utilize these tools but the important item to remember is that data is moving through multiple tiers. Its security must be accounted for on each level.

The primary two methods of solving the heterogeneous data dilemma to date are using encryption at a customer level and using multiple database instances.

3.1 Encryption by Customer

In the best practice three-tiered architecture, encryption keys are commonly stored on the application tier. One segregation method which has proven successful is to provide a different key for each customer. An accidental cross-pollination of data due to code

defect would result in the decryption of data with a key that does not match. It is necessary to retain the keys outside the database tier to add integrity to the process with "Defense in Depth". Defense in Depth is practical security strategy for achieving Information Assurance in today's highly networked environments. It is a "best practices" strategy in that it relies on the intelligent application of techniques and technologies that exist today³. The goal is to introduce barriers at every possible level to prevent a breach and to not rely on a single tier or device to prevent unauthorized access. Accordingly, leaving the key with the data would be negligent. Distinct keys should be utilized for Web to browser cookie encryption in the same way that data being stored on the disk is encrypted.

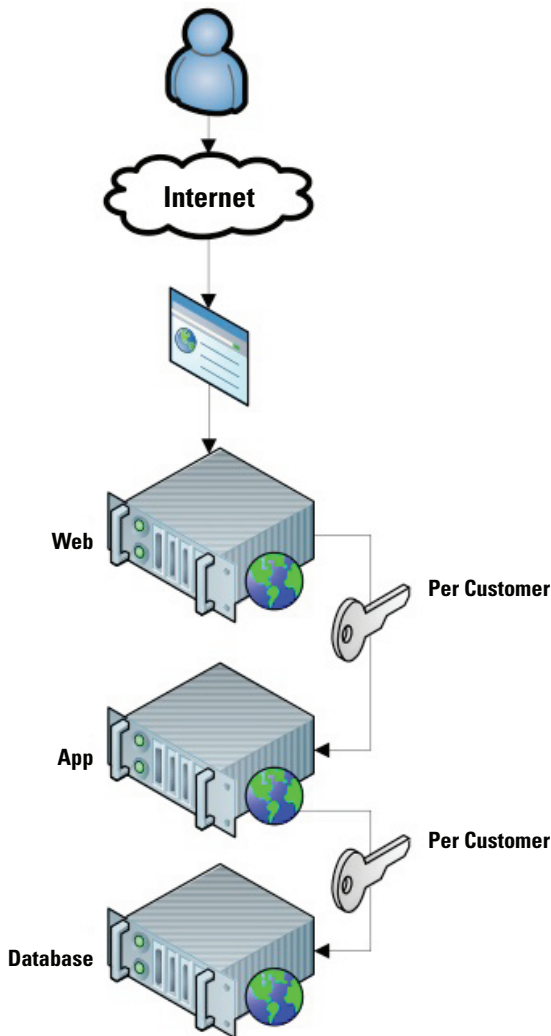


Figure 3

Figure 3 illustrates these concepts. There are separate keys for the Web

to application tier communication and the application to database tier communication. The later key is used to encrypt data upon the database server disk.

Consider then, in Figure 4, what the database and application tiers will look like internally. Each customer is provided a unique set of encryption keys for both the top and bottom tiers. There would be a need for a key which connects from the application tier to the database tier if the database is being used to determine who the incoming customer is.

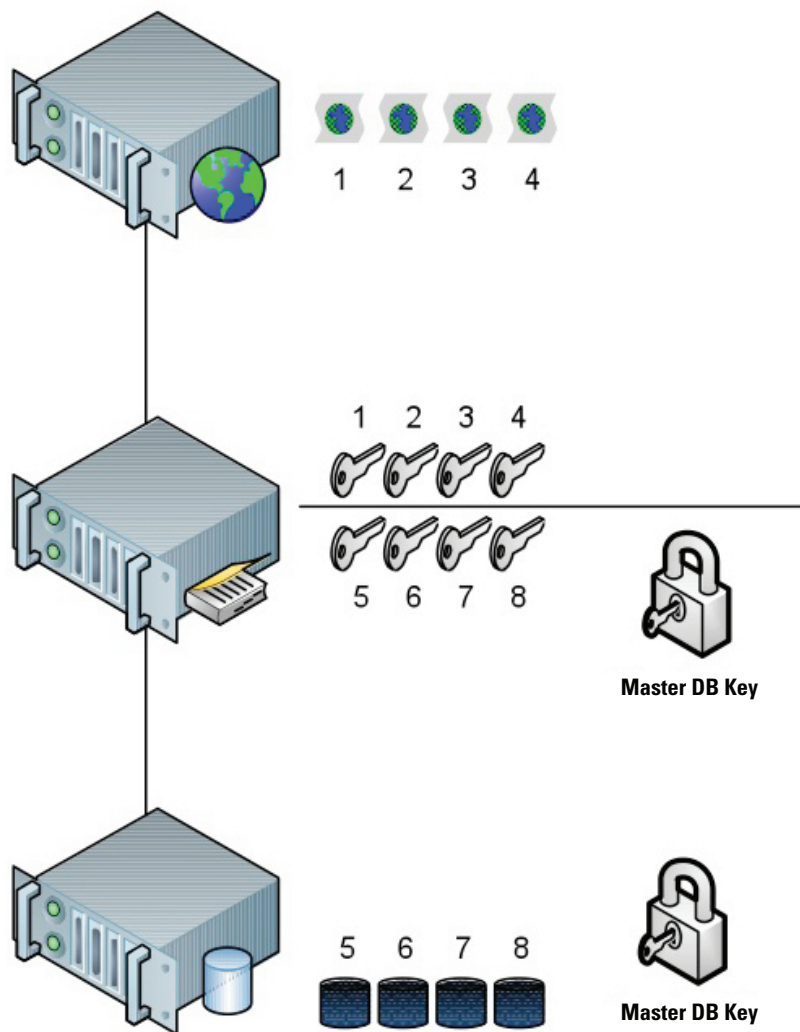


Figure 4

As a customer comes to the top of the SaaS stack, you will need to identify them and which database they should be accessing. The Master DB Key concept simply opens a database linking access method (URL, URL parameter, inbound IP etc) to a specific customer so that the code is able to choose the correct key.



3.2 Database Instances

Multiple database instances are another potential solution for keeping data segregated. However, since the overall goal of SaaS is to minimize overall system maintenance and implementation costs, the encryption method is preferable. If you do choose this method, be prepared to develop a custom interface to automate the management of this methodology. Utilizing standard database tools will increase maintenance and deployment time but this can often be mitigated by automating those two processes.

4.0 DATA TRANSMISSION

The concept of multi-tenancy lends itself to general Internet access and not to the use of VPNs or private lines. The very essence of SaaS is the concept of multi-tenancy. The use of distinct communication channels for each customer is counterintuitive, costly and can become a maintenance nightmare. The use of private lines or VPNs should be avoided. Transmission security should be designed into the system such as SSL. Because of its use of the public Internet, movement of data from the SaaS platform to the customer is nearly identical to the security challenges faced by other business models, such as the ASP business model. ASP and SaaS use the unsecured public Internet to transmit data. These security challenges are addressed by embracing the following three primary concepts:

- > Confidentiality — Data can only be viewed by the intended recipient.
- > Integrity — Data cannot be altered without detection.
- > Non-repudiation — Recipient of data has proof that the data originated from the SaaS provider.

These three concepts can be addressed as a group because a solution to one should include a comprehensive solution to all three. An obvious tool in this effort is the SSL or TLS encryption handled between Web server and browser but that alone is not enough to ensure all three of our concepts are addressed. These cryptographic protocols are designed specifically to prevent eavesdropping, tampering and forgery. It is strongly recommended that weak ciphers and algorithms be disabled on the Web server. TLS 1.0 and above or SSL 3.0 or above are recommended. SSL 2.0 has several flaws but is still support by browsers such as IE 6. However, this is an area where there is no perfect solution and a “defense-in-depth” strategy is necessary. Here are a few guidelines:

- > If data is being sent in a trusted environment⁴, the keys should be rotated frequently (at the very least, annually).
- > An automated process should be set up to scan the audit logs daily. High priority items should be hand reviewed immediately.
- > If handling documents, use Adobe PDF format which has numerous security features available which are not available in HTML or Microsoft Word Documents.



5.0 PHYSICAL SECURITY

In general a SaaS company is a software company and a services company. Security must be layered into the application but physical security is just as important. Often, physical security is outsourced to a third party datacenter who is better able to specialize in this area. Just as a programmer should not have to fix the power drops, they cannot be expected to maintain physical barriers. The location of the hosting should be a selling point for the SaaS provider. A SAS70 level II provider is generally quite easy to find but many data centers are now stepping up to certification by more rigorous standards such as PCI⁵. A hosting provider needs to be carefully chosen.

6.0 ADDITIONAL SECURITY CONSIDERATIONS

There are a few additional items which have grown in importance over the past few years as more companies rely on third parties for their mission critical activities or sensitive data. These are addressed here.

6.1 Audit Trail

At all times it is necessary to maintain electronic audit evidence (EAE) in order to satisfy any legal proceedings one of your customers may face. It is important to maintain records demonstrating proof of origin, all alterations/additions/deletions and approvals where appropriate. This may not be as vital in the case of storing records on pet care, for example, but even the ubiquitous credit card transaction immediately brings in the PCI compliance documentation requiring this unalterable logging mechanism. This is a case of better safe than sorry. The audit log should be encrypted and kept on a network segment which your normal system engineers do not have access in order to maintain the log's viability.⁶

6.2 Disclosure of Provider's Proprietary Information

Where in-house systems are able to rely on the secrecy of the application architecture as one layer of security, SaaS providers are often called upon by their customers to provide proprietary information about the internal workings of their application. When providing information, be sure to have engineers sanitize it and have it approved by the CIO/CTO. This would include removing IP information, removing brand names where possible, aggregating information about specific farms into "Web server farm" etc. Although it may not be possible to completely sanitize this data, it should certainly be sanitized to the best of one's ability and double-checked by a C-level officer. A time-saving tip here it to build a sanitized version for general distribution to any customer who has already signed an NDA.



6.3 AJAX

AJAX has become ubiquitous in today's Web based applications. While it does provide amazing new functionality it is also important to remember that it is increasing the attack surface. With AJAX, calls from outside the normal Web pages will inevitably be made to the Web service that is being called. This has introduced the new XSS Cross scripting attack. It is strongly recommended that every call is checked to the Web server and the Web service for authentication and authorization before proceeding with the request. There should also be awareness of this vulnerability when performing QA or penetration testing. All business logic should be kept on the server, all data validated, there should be checks for SQL or JavaScript injection attacks and, most importantly, authentication should be verified.

6.4 Multifactor Authentication

Authentication options have improved over time. Multi-factor authentication in highly sensitive environments should be strongly considered. Multifactor authentication refers to the use of more than just password to prove your identity. Who you are, what you know and what you have are the three basic pillars of this. Multi-factor authentication is superior to standard password authentication because it requires, for example, biometrics or a dongle to authenticate the user.


Multifactor authentication should be consider an absolute must for administrators who are unable to be on-site but must access the production environment.

6.5 Patches

It is an important security practice to proactively receive notifications from all vendors about security vulnerabilities and apply them appropriately. For "critical" patches it may be necessary to apply the patch same day even if that means down time. Generally, regulatory environments such as PCI permit you 30 days for patching non-critical issues. With the use of an ISV, the effort and risk around this can be dramatically reduced.

6.6 Generation of Keys

The concept of generating a key is the concept of generating entropy and keeping keys from being available to an individual employee. With entropy, you are looking to create the key in a manner unlikely to be reproduced. There are services available to perform this function. It is highly recommended that a key generator is not used that creates a key based on a passphrase someone has entered. This makes it far easier for someone to decrypt the data. As for separation of the visibility of the key, ideally a system is implemented within the SaaS offering for providing and implementing these keys into a secure place within the stack (most likely the application tier) which does not require an individual touches the key. The



only other option is to build into the software the ability to have two people enter half a key and have the system store it securely. This method is far more complex and the former is a preferred method.

7.0 SUMMARY

Security is one of the most important components of any successful SaaS offering. It is core and must be considered and integrated from day one of architecting a SaaS application. It cannot be added on after the fact. Here are a few key components to remember:

- > Utilize “defense-in-depth” to make sure your security never relies on one single component.
- > When in doubt, use encryption and when using encryption enforce strong ciphers and algorithms.
- > Rotate your keys frequently.
- > Always monitor your logs.
- > Consider your system’s physical security.
- > Choose the identity management methodologies carefully.
- > Segregate identity management from authorization.

8.0 REFERENCES

¹<http://www.oasis-open.org> — The home page of the SAML open standard. Security Assertion Markup Language.

²<http://shibboleth.internet2.edu/>

³For an overview of the Defense in Depth strategy, see the NSA at <http://www.nsa.gov/snac/support/defenseindepth.pdf>

⁴In this context, we use “trusted environment” to mean an environment where two parties (the SaaS provider and their customer) have exchanged keys known only to each other.

⁵<https://www.pcisecuritystandards.org/>

⁶Note the FBI’s Information Technology Strategic Plan Synopsis lists its #5 challenge as “Information Assurance Balancing Security and Convenience.” The plan requires a “defense in depth” approach and a Single Sign-on Public Key Infrastructure. <http://www.fbi.gov/hq/ocio/documents/itsp.pdf>

9.0 GLOSSARY OF TERMS

Term	Description
Authentication	Proving the identity of a user.
Authorization	Determining what resources or actions a user is permitted to utilize.
Information Assurance	Information Assurance is achieved when information and information systems are protected against attacks through the application of security components such as: Availability, Integrity, Authentication, Confidentiality, and Non-Repudiation.
Defense In Depth	The use of multiple defense mechanisms, layered upon each other to provide better protection than any single instrument.
Trust	A high degree of assurance that both the SaaS provider and the consumer system are meeting security requirements.



Worldwide Headquarters

Progress Software Corporation, 14 Oak Park, Bedford, MA 01730 USA
Tel: +1 781 280-4000 Fax: +1 781 280-4095
www.progress.com

For regional international office locations and contact information, please refer to www.progress.com/worldwide

© Copyright 2008 Progress Software Corporation. All rights reserved. Progress is a registered trademark of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. or other countries. Any other trademarks contained herein are the property of their respective owners.