Microsoft

Security Best Practices For Developing Windows Azure Applications

Authors
Andrew Marshall (Senior Security Program Manager, Security Engineering)
Michael Howard (Principal Security Program Manager, Security Engineering)
Grant Bugher (Lead Security Program Manager, OSSC)
Brian Harden (Security Architect, OSSC)

Contributors
Charlie Kaufman (Principal Architect)
Martin Rues (Director, OSSC)
Vittorio Bertocci (Senior Technical Evangelist, Developer and Platform Evangelism)

June 2010 (REVISION 2)

Table of Contents

Executive Summary

As businesses seek to cost-effectively consume IT services, interest is growing in moving computation and storage from on-premise equipment to Internet-based systems, often referred to as "the cloud."

Cloud computing is not restricted to large enterprises; small companies benefit greatly from moving computing and storage resources to systems such as Windows Azure. In fact, smaller companies are adopting this new paradigm faster than larger companies[1].

The idea that purchasing services from a cloud service provider may allow businesses to save money while they focus on their core business is an enticing proposition. Many analysts view the emerging possibilities for pricing and delivering services online as disruptive to market conditions. Market studies and the ensuing dialogue among prospective customers and service providers reveal some consistent themes and potential barriers to the rapid adoption of cloud services. Business decision makers want to know, for example, how to address key issues of security, privacy and reliability in the Microsoft Cloud Computing environment, and they are concerned as well about the implications of cloud services for their risk and operations decisions.

This paper focuses on the security challenges and recommended approaches to design and develop more secure applications for Microsoft's Windows Azure platform. Microsoft Security Engineering Center (MSEC) and Microsoft's Online Services Security & Compliance (OSSC) team have partnered with the Windows Azure team to build on the same security principles and processes that Microsoft has developed through years of experience managing security risks in traditional development and operating environments.

Intended Audience

This paper is intended to be a resource for technical software audiences: software designers, architects, developers and testers who design, build and deploy more secure Windows Azure solutions.

This paper is organized into two sections[2]:

- Overview of Windows Azure security-related platform services; and
- Best practices for secure design, development and deployment:
    - Service-layer/application security considerations
    - Protections provided by the Azure platform and underlying network infrastructure.
    - Sample design patterns for hardened/reduced-privilege services.

Overview of Windows Azure security-related platform services

The following sections describe some of the platform services and security functionality available to developers who build applications on Windows Azure.

---

[1] "Cloud Computing: Small Companies Take Flight" BusinessWeek http://www.businessweek.com/technology/content/aug2008/tc2008083_619516.htm

[2] The distinction between "security features" and "secure features" is an important one. "Security features" are the technologies, such as authentication or encryption that can help protect a system and its data. "Secure features" are technologies that are resilient to attack, such as encryption key storage and management or code with no known vulnerabilities

Identity Management and Access Control

Identity management has emerged as an increasingly complex issue for developers, and proper identity management remains a priority, even as business networks change. Identity management is as much about preventing unauthorized third-party access to data as it is about controlling the authorized use of data. Identity management helps systems control the amount and type of data that users can access, and it helps ensure that users are performing necessary functions at the lowest-possible privilege levels. Identity management is also critical for maintaining separation of roles and duties, which may be required by specific regulatory and compliance standards.

Today, developers face a common challenge: To provide authorized users, clients and systems with access to the data they require at any time, from any technology, in any location, while meeting basic security requirements for confidentiality, availability and integrity. At first glance, cloud computing technology might seem inappropriate or poorly suited to meet such rigorous standards: Developers seek to restrict access to data that exists in a comparatively uncontrolled environment (cloud), and that data may be co-mingled with resources that are owned by someone else. Those concerns, while legitimate, can be effectively addressed by using claims-based identity.

Claims-based identity, an approach to authentication and access management based on open protocols, is an access control strategy that is consistently applied across the full range of Microsoft product and services. One of the key properties of claims-based identity is that it reduces infrastructure dependencies because applications protected with claims-based identity can be hosted on-premises or in the cloud without changes. Applications targeting Windows Azure can take advantage of the same developer tools, identity management features and services that are available to their on-premises counterparts.

Below is a list of the most relevant identity technologies and services that can work with Windows Azure to protect applications and resources.

- Windows Identity Foundation
- Active Directory Federation Services 2.0
- Windows Azure AppFabric Access Control Service

Windows Identity Foundation

Windows Identity Foundation (WIF) is the latest addition to the foundational technologies in the .NET Framework. It enables .NET developers to offload the identity logic from their application, providing a solid development model based on separation of concerns. Non-experts can easily secure their applications without being exposed to the underlying complexity of cryptography and protocols, leveraging Visual Studio integration features such as point-and-click wizards which result in applications protected using open, interoperable standards such as WS-Federation and WS-Trust.

Despite the easy to use programming model, which unifies ASP.NET web applications and (Windows Communication Foundation) WCF SOAP services under a single object model, Windows Identity Foundation has a full range of security of features offered by WS-Security, the SAML token format and many other enterprise-grade industry standards.

When using Windows Identity Foundation the mechanics of authentication are provided by external services, using platform-independent protocols. The application receives information about authenticated users in forms of claims, which can be used for simple or traditional role-base access control (RBAC) to sophisticated access control policies.

Because open standards are used, the authentication can take place regardless of where the user accounts are maintained or where the application is hosted: as a result, single sign on (SSO) across on-premises and Windows Azure hosted resources is easily achieved.

Although the authentication services can be provided from any platform complying with the open protocols used by Windows Identity Foundation, the best way to leverage existing investments in the Windows infrastructure is to outsource authentication to Active Directory Federation Services 2.0.

Active Directory Federation Services 2.0

Although Active Directory Federation Services 2.0 (AD FS 2.0) is a technology that can be deployed on-premises, it can play a key role in enabling authentication for Windows Azure applications.

AD FS 2.0 is a Windows Server role that extends Active Directory (AD) with claims-based identity capabilities. AD FS 2.0 provides AD with a Security Token Service (STS) which is a single interface enabling existing users to authenticate using applications regardless of whether they are hosted in a data center, at one partner's site, or in the cloud. Users are no longer constrained by the boundaries of their local network: if an application hosted in Windows Azure has been developed using Windows Identity Foundation (or an equivalent stack complying with the same open standards), AD FS 2.0 allows instantly granting anybody with an account in the local directory access to this application. All without requiring any form of synchronization, new account provisioning or duplication.

AD FS 2.0 facilitates the establishment and maintenance of trust relationships with federated partners, simplifying access to resources and distributed single sign-on.

AD FS 2.0 implements standards such as WS-Trust, WS-Federation and the SAML protocol, and successfully passed the latest public Liberty Alliance SAML 2.0 interoperability testing which proved out-of-the-box interoperability with products from IBM, Novell, Ping Identity, SAP, Siemens and many others.

Windows Azure Platform AppFabric Access Control Service

The Windows Azure platform AppFabric Access Control (AC) service is a hosted service that provides federated authentication and rules-driven, claims-based authorization for REST Web services. REST Web services can rely on AC for simple username/password scenarios, in addition to enterprise integration scenarios that use Active Directory Federation Services 2.0.

Applications exposing REST Web services can take advantage of the AC regardless of where they are deployed, either on-premises, in Windows Azure, or anywhere else where an internet connection is available. AC allows customers to achieve true externalization of authorization policies, offering the chance of decoupling applications from most of their authorization logic by hosting it (in the form of claims transformation rules) at the AC itself.

The AC leverages the OAuth Web Resource Authorization Protocol (OAuth WRAP), a lightweight protocol which makes it possible to take advantage of claims-based identity with REST-based APIs, without imposing strong requirements on clients and service providers: this enables unprecedented reach, enabling a wide array of device types and communication stacks to participate in secure transactions. OAuth WRAP is the basis for the upcoming Oauth 2.0 specification, a protocol which is catalyzing the consensus of the key players in the Web space.

AC is also capable of bridging the enterprise identity and the REST worlds, thanks to its capability of using SAML tokens issued by an AD FS 2.0 instance for accessing REST services via OAuth WRAP protocol.

Future releases of the AC will, among many others, support the protocols implemented by WIF and ADFS 2.0 natively, ensuring seamless integration across solutions based on those technologies

Designing More Secure Windows Azure Services

When it comes to cloud-based solutions, it is more important for software designers and developers to anticipate threats at design time than is the case with traditional boxed-product software deployed on servers in a corporate datacenter. This section highlights some specific threats that developers are responsible for mitigating in the cloud and describes the protections that Windows Azure provides against an array of service-, platform- and infrastructure-layer security threats.

Windows Azure Service-layer security considerations

The threat landscape for a cloud-based web service is substantially different from traditional hosted web services in terms of mitigating tools and technologies. Threats also vary dramatically among cloud providers. Appendix B contains a security threat matrix specific to Windows Azure. This matrix can be used to help identify which security threats are most important for an application.

The key points are as follows:

- Developers must map the traditional on-premise enterprise security requirements of their application or service to Windows Azure platform services that provide comparable functionality. Any remaining threats must be mitigated by the application or service. Consult Appendix B for details about specific mitigations.

- Understand the security requirements of the service being designed or migrated, especially in the context of authentication, authorization and auditing. The platform services which provide these features (Windows Identity Foundation, Windows Azure AppFabric Access Control Service, Windows Azure Monitoring and Diagnostic APIs) and the methods developers use to invoke them are substantially different from those provided in an on-premise enterprise deployment (Kerberos, Active Directory, Windows Event Logs).

- Leverage Windows Azure platform services to build more secure applications.

Despite some of the protections that moving to the cloud may offer, developers are still responsible for writing good code -- and they are still responsible for the security of their applications when dealing with threats to the web service code itself. Input field constraint, sanitization and validation are still the most important ways to help protect a web service application, whether it is a cloud-based application or not. Windows Azure runs web roles in Internet Information Services 7 Hosted Web Core. Because of IIS7's secure default configuration, developers inherit some basic IIS7 protections, such as the ValidateRequest configuration setting.

Developers must mitigate cross-site scripting attacks by encoding and validating inputs to their services with the Microsoft Anti-Cross-Site-Scripting library or other similar libraries, especially if that input is displayed back to the user in a web page. Cross-site request forgery (CSRF) attacks must also be mitigated by setting a hidden session token in client requests and setting Page.ViewStateUserKey to the current session ID.

Namespace Configuration Issues

Here are a few more configuration issues to be aware of in the cloud:

- Avoid using *servicename*.cloudapp.net domain name - use a custom domain name instead. An important distinction between the enterprise and the cloud is that the cloudapp.net namespace is

shared among all Azure customers while the namespace Microsoft.com is wholly-owned and controlled by Microsoft. This means that the cloudapp.net namespace is inherently less trusted than the domain namespace of a single enterprise because one customer of Windows Azure does not automatically trust all other customers in that domain namespace. Don't create code that requires users to place cloudapp.net in the trusted sites list in their web browser.

- Never scope cookies or document.domain to cloudapp.net. Instead, scope to the service subdomain (such as contoso.cloudapp.net, for example or, better, www.contoso.com).

For most content, only interactions with content from the same domain are allowed. For example, a typical page on www.microsoft.com can freely script content on any other page on www.microsoft.com, but it cannot script to pages that are located on a different Web domain. The DHTML Object Model uses the document.domain property to enforce this restriction. Only pages with identical domain properties are allowed free interaction. The protocol of the URL must also match. For example, an HTTP page cannot access HTTPS content.

Important: Any attempt to broaden document.domain access can leave a service open to scripting attacks from the entire cloudapp.net domain namespace. IIS7 sets document.domain to the full subdomain by default (like contoso.cloudapp.net or www.microsoft.com), but the scoping is so often changed by web developers that it warrants mention.

Data Security

When designing Web Role interactions with Windows Azure Storage, Shared Access Signatures can be a powerful tool. Shared Access Signatures are effectively access tokens that bestow a set of rights against a blob or blob container that normally would not be set public. Since web applications are responsible for generating these tokens, they are also responsible for securely distributing them. In the event that one of these tokens is compromised, developers can either update the blob/container metadata to invalidate the token, or simply create a new container for the blob data. However, this is still a reactive measure taken after damage could have already been done. Outlined, below are guidelines for minimizing risk while using Shared Access Signatures.

- Generate Shared Access Signatures with the most restrictive set of ACLs possible that still grant the access required by the trusted party.
- Use the shortest lifetime possible.
- Use HTTPS in the request URL so the token cannot be snooped on the wire.
- Remember that these tokens are only used for temporary access to non-public blob storage – as with passwords, it's a bad idea to use the same ones over and over.

Windows Azure table storage provides a simple structured storage environment and is not SQL-based, so common SQL injection vulnerabilities do not apply to an application that uses it. However, applications using SQL Azure or other relational database services still need to mitigate traditional SQL injection threats.

Table storage requests are either made directly, using HTTP GET and POST requests, or they are translated into these requests by the SDK and LINQ. Since HTTP requests are text, if this request string were constructed based on data gathered from users, then the user's strings could possibly change the semantics of the request (e.g. by inserting carriage returns or & characters.) To avoid injection attacks, do not base any container names, blob names, blocks, block IDs, or table names on data gathered from users. When constructing a REST

query that involves user data, help ensure the data's safety by URL encoding before concatenating it into the query.

Handling Secret Information

Currently there is no support for Data Protection API (DPAPI)-like persistence of secret data in Windows Azure Storage. If a service must encrypt secret data at rest within Windows Azure Storage, then it needs to encrypt that data offsite, and before uploading the encrypted payload to blob storage. This can be done easily with an AES key generated on a client machine or elsewhere within the enterprise. Also, developers should not upload the key or any keying material to Windows Azure Storage, regardless of how careful they are about hiding it. If any computer or storage services were compromised, it could lead to encryption keys being exposed. Microsoft recommends using 256-bit AES keys for symmetric encryption.

Developers also should not store private keys associated with SSL/TLS certificates in Windows Azure Storage. Instead, upload them through the Developer Portal and access them via thumbprint references in the Service Configuration. Windows Azure will not only store these certificates encrypted at all times, but also securely provision them into the certificate stores of the service's web roles upon boot. Developers should not attempt to store certificates anywhere on their own as these actions would constitute re-inventing a protection already supplied by the platform.

Sample code that shows how to install certificates in Windows Azure is available at http://blogs.msdn.com/jnak/archive/2010/01/29/installing-certificates-in-windows-azure-vms.aspx.

Appendix D contains security best practices for cryptography usage.

Auditing and Logging

Local VM disk access by Windows Azure roles should be considered temporary and not reliable for anything more than temp files and caching. On Windows Azure, events are not written to application, security, or audit event logs as they are on Windows servers. Instead, events data is logged to Windows Azure Storage through the Monitoring and Diagnostics Agent via trace listeners in web application code. The logs are then stored long-term in Windows Azure Storage via scheduled transfers performed by the Monitoring Agent.

Windows Azure Table Storage is used to log certain types of events, including Windows Event and Diagnostics Logs, Windows Azure Logs and performance counters.

Windows Azure does not currently support encryption-at-rest of table storage information, so developers should not write sensitive information to any events stored in Windows Azure Table Storage. Refer to "Handling Secret Information" on page 8 for more information about encrypting sensitive data.

Developers must decide to use HTTP or HTTPS depending on the contents of the logs. If an application is logging a large amount of data that won't be of interest to outside parties or eavesdroppers, then HTTP can be used for a faster transfer. However, Microsoft recommends protecting all log data in transit to Windows Azure Storage by using HTTPS.

Request Throttling / Input Sanitization

Developers must do application-level throttling of incoming requests for any kind of complex, time-intensive operation.

It is also important to fuzz test any new parsers deployed as part of a web service. The Microsoft Security Development Lifecycle (SDL) portal at http://www.microsoft.com/sdl provides resources on fuzzing parsers. If a service is parsing a proprietary file or request format (perhaps encapsulated inside HTTP), then fuzz test it to ensure the code can correctly accommodate malformed input. Generally speaking, an automated fuzzing

framework that fuzzes 100,000 iterations of each new format or protocol without crashing, or runs non-stop for 24 hours, will give developers a good indication of threat resistance. This is the fuzzing requirement that Microsoft currently applies to "boxed-product" software.

Windows Azure Platform- & Infrastructure-layer security protections

This section outlines security threats that are mitigated on developers' behalf by the Windows Azure platform and underlying network infrastructure.

Port Scanning/ Service Enumeration

The only ports open and addressable (internally or externally) on a Windows Azure VM are those explicitly defined in the Service Definition file. Windows Firewall is enabled on each VM in addition to enhanced VM switch packet filtering, which blocks unauthorized traffic

Denial of Service

Windows Azure's load balancing will partially mitigate Denial of Service attacks from the Internet and internal networks. This mitigation is done in conjunction with the developer defining an appropriate Service Definition VM instance count scale-out. On the Internet, Windows Azure VMs are only accessible through public Virtual IP Addresses (VIPs). VIP traffic is routed through Windows Azure's load-balancing infrastructure. Windows Azure monitors and detects internally initiated Denial of Service attacks and removes offending VMs/accounts from the network. As a further protection, the root host OS that controls guest VMs in the cloud is not directly addressable internally by other tenants on the Windows Azure network and the root host OS is not externally addressable.

Windows Azure is also reviewing additional Distributed Denial of Service (DDoS) solutions available from Microsoft Global Foundation Services to help further protect against Denial of Service attacks.

Spoofing

VLANs are used to partition the internal network and segment it in a way that prevents compromised nodes from impersonating trusted systems such as the Fabric Controller. At the Hypervisor VM Switch, additional filters are in place to block broadcast and multicast traffic, with the exception of what is needed to maintain DHCP leases. Furthermore, the channel used by the Root OS to communicate with the Fabric Controller is encrypted and mutually authenticated over an HTTPS connection, and it provides a secure transfer path for configuration and certificate information that cannot be intercepted.

Eavesdropping / Packet Sniffing

The Hypervisor's Virtual Switch prevents sniffer-based attacks against other VMs on the same physical host. Top-of-rack switches will be used to restrict which IP and MAC addresses can be used by the VMs and therefore mitigate spoofing attacks on internal networks. To sniff the wire inside the Windows Azure cloud environment, an attacker would first need to compromise a VM tenant in a way that elevated the attacker to an administrator on the VM, then use a vulnerability in the hypervisor to break into the physical machine root OS and obtain system account privileges. At that point the attacker would only be able to see traffic inbound to the compromised host destined for the dynamic IP addresses of the VM guests controlled by the hypervisor.

Multi-tenant hosting and side-channel attacks

Information disclosure attacks (such as sniffing) are less severe than other forms of attack inside the Windows Azure datacenter because virtual machines are inherently untrusted by the Root OS Hypervisor. Microsoft has done a great deal of analysis to determine susceptibility to side-channel attacks. Timing attacks are the most difficult to mitigate. With timing attacks, an application carefully measures how long it takes some operations to complete and infers what is happening on another processor. By detecting cache misses, an attacker can figure out which cache lines are being accessed in code. With certain crypto implementations involving lookups

from large tables, knowing the pattern of memory accesses - even at the granularity of cache lines - can reveal the key being used for encryption. While seemingly far-fetched, such attacks have been demonstrated under controlled conditions.

There are a number of reasons why side-channel attacks are unlikely to succeed in Windows Azure:

- An attack works best in the context of hyper-threading, where the two threads share all of their caches. Many current CPUs implement fully independent cores, each with a substantial private cache. The CPU chips that Windows Azure runs on today have four cores per chip and share caches only in the third tier.

- Windows Azure runs on nodes containing pairs of quad-core CPUs, so there are three other CPUs sharing the cache, and seven CPUs sharing the memory bus. This level of sharing leads to a great deal of noise in any signal from one CPU to another because actions of multiple CPUs tend to obfuscate the signal.

- Windows Azure generally dedicates CPUs to particular VMs. Any system that takes advantage of the fact that few servers keep their CPUs busy all the time, and implements more logical CPUs than physical CPUs, might open the possibility of context switches exposing cache access patterns. Windows Azure operates differently. VMs can migrate from one CPU to another, but are unlikely to do so frequently enough to offer an attacker any information.

External Verification

Microsoft contracted two top-tier penetration testing firms to conduct security assessments on different elements of the Windows Azure architecture before the Professional Developers Conferences (PDC) in 2008 and 2009. Each firm spent a significant amount of time examining hardened virtualization boundaries and probing for side-channel or I/O thrashing attacks. Neither firm was able to mount a successful attack against this design. That is not to say that such attacks are impossible to execute successfully, but six professional penetration testers working over the course of seven weeks were unable to do so. These tests supplemented required internal security testing.

Runtime Security: Role Separation and Process privileges in Full Trust vs. Windows Azure Partial Trust

Windows Azure has a custom, restricted-privilege trust model available to all roles called "Windows Azure Partial Trust." Based on customer demand, Windows Azure also supports Full Trust with Native Code Execution.

Full Trust with Native Code Execution facilitates the following scenarios:

- Use of FastCGI or PHP.

- Migration of traditional web services to the cloud.

- Role invocation and spawning Windows sub-processes (native code or managed).

- Calls into native libraries via P/Invoke (Platform Invocation Services).

A role that does not require the above functionality should have Windows Azure Partial Trust enabled. This option not only minimizes the attack footprint of the role in question, but it also helps reduce damage caused by a compromised role (see the Windows Azure Partial Trust comparison to Full Trust later in this document).

Regardless of the trust model selected, customer roles are hosted in non-Admin svchost processes. These svchost processes run under virtual service account SIDs, and not as local system or network service or administrator. This limitation provides two levels of defense-in-depth protection:

- A compromised service cannot manipulate other instances of that service if those instances reside on other VMs, and that helps limit an intrusion to a machine targeted for attack.

- A compromised service cannot easily attack the VM guest operating system in order to compromise the physical host OS. A compromised web role running in Windows Azure Partial Trust cannot P/Invoke native code binaries which could then be used to expose sensitive information about the service from the virtual machine (such as storage key data or application-specific intellectual property that could be present in memory). Direct attacks on the root host through the root/guest Hypervisor boundary in an attempt to gain control of the system are also mitigated.

Putting it all together: Creating more secure Windows Azure applications

By understanding the Windows Azure runtime trust models and the security protections and responsibilities of each cloud layer, developers can build hardened applications using the design best practices outlined below.

Isolate web roles and separate duties of individual roles in order to maximize the use of Windows Azure Partial Trust.

The Trust Levels Appendix C illustrates a subset of Partial Trust Policy Restrictions. The complete listing of restrictions is available in "Windows Azure Partial Trust Policy Reference." The impact of these security restrictions illustrates the value of using Windows Azure Partial Trust to help secure Windows Azure services, especially externally-facing web roles. Most access to the local environment variables, file system and registry is restricted. Socket and Web connection permissions are also locked down.

Use the "Gatekeeper" design pattern to separate role duties and isolate privileged access

A Gatekeeper is a design pattern in which access to storage is brokered so as to minimize the attack surface of privileged roles by limiting their interaction to communication over private internal channels and only to other web/worker roles. These roles are deployed on separate VMs. In the event of a successful attack on a web role, privileged key material is not compromised. The pattern can best be illustrated by the following example which uses two roles:

- The GateKeeper – this is a web role that services requests from the Internet. Since these requests are potentially malicious, the Gatekeeper is not trusted with any duties other than validating the input it receives. The GateKeeper is implemented in managed code and runs with Windows Azure Partial Trust. The service configuration settings for this role do not contain any Shared Key information for use with Windows Azure Storage.

- The KeyMaster – this is a privileged backend worker role that only takes inputs from the Gatekeeper and does so over a secured channel (an internal endpoint, or queue storage – either of which can be secured with HTTPS). The KeyMaster handles storage requests fed to it by the GateKeeper, and assumes that the requests have been sanitized to some degree. The KeyMaster, as the name implies, is configured with Windows Azure Storage account information from the service configuration to enableretrieval of data from Blob or Table storage. Data can then be relayed back to the requesting

client. Nothing about this design requires Full Trust or Native Code, but it offers the flexibility of running the KeyMaster in a higher privilege level if necessary.

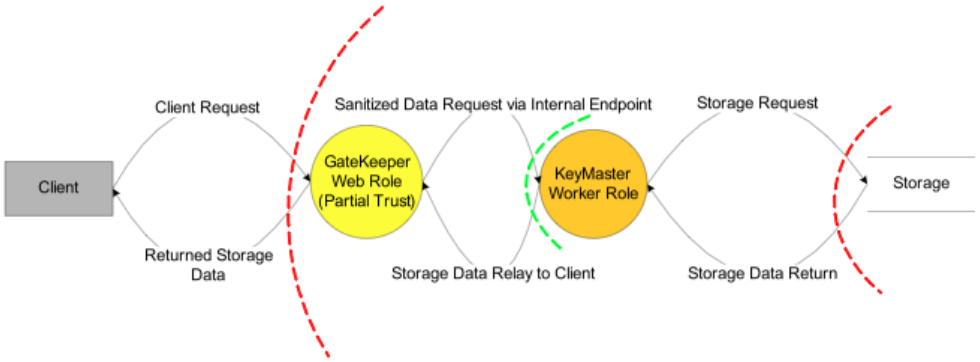

Figure 1: The Gatekeeper Design Pattern

This solution is not ideal, since the KeyMaster is relying on GateKeeper to tell it what content to serve out. However, it does provide for separation of duty and it can poses additional challenges for an attacker. The amount of trust the KeyMaster places in the GateKeeper can be custom-tailored, but it is advisable to minimize the types of access requests allowed from the GateKeeper -- allowing the KeyMaster to accept Storage Blob API read and list operations from GateKeeper, but not add or delete operations, for example.

This pattern can easily be adapted to work with more complex service architectures on Windows Azure. All that is needed is to put a partial trust "Gatekeeper" in front of more privileged (or native code) roles to do the parsing of potentially malicious data from the network. The concept is analogous to a firewall running as Network Service to interact with foreign TCP payloads before pushing them up the stack to more critical components.

Use multiple storage keys to restrict access to privileged information where the Gatekeeper pattern does not apply.

In scenarios where a partial-trust Gatekeeper cannot be placed in front of a full-trust role, a multi-key design pattern can be used to protect trusted storage data. An example case of this scenario might be when a PHP web role is acting as a front-end web role, and placing a partial trust Gatekeeper in front of it may degrade performance to an unacceptable level.

Up to five storage keys can be assigned to one Windows Azure subscription. This diversity can be used to minimize exposure of a particular key to theft by placing lower-trust keys on lower-trust roles and higher-trust keys on higher-trust roles.

"Untrusted web role A" in the figure below has access to only one storage account, and it is not trusted to do anything but parse incoming requests and log them to storage account 'A'. "Trusted Role B" (in the background) has the account keys for both storage 'A' and 'B', but is not exposed to potentially-malicious, malformed inputs. "Trusted Role B" processes and filters requests from the untrusted role via storage 'A,' but expects a specific structure or format. 'B' never fully trusts 'A,' allowing for the "real" storage key of value (account 'B') to be protected even in the event of a fully compromised, externally-facing, untrusted role. As with the Gatekeeper pattern, the trusted role should not provide unfettered access to the trusted store on behalf of Web Role 'A.' Instead, only a subset of legitimate commands should be serviced (such as read, but not create/update/delete in certain cases).

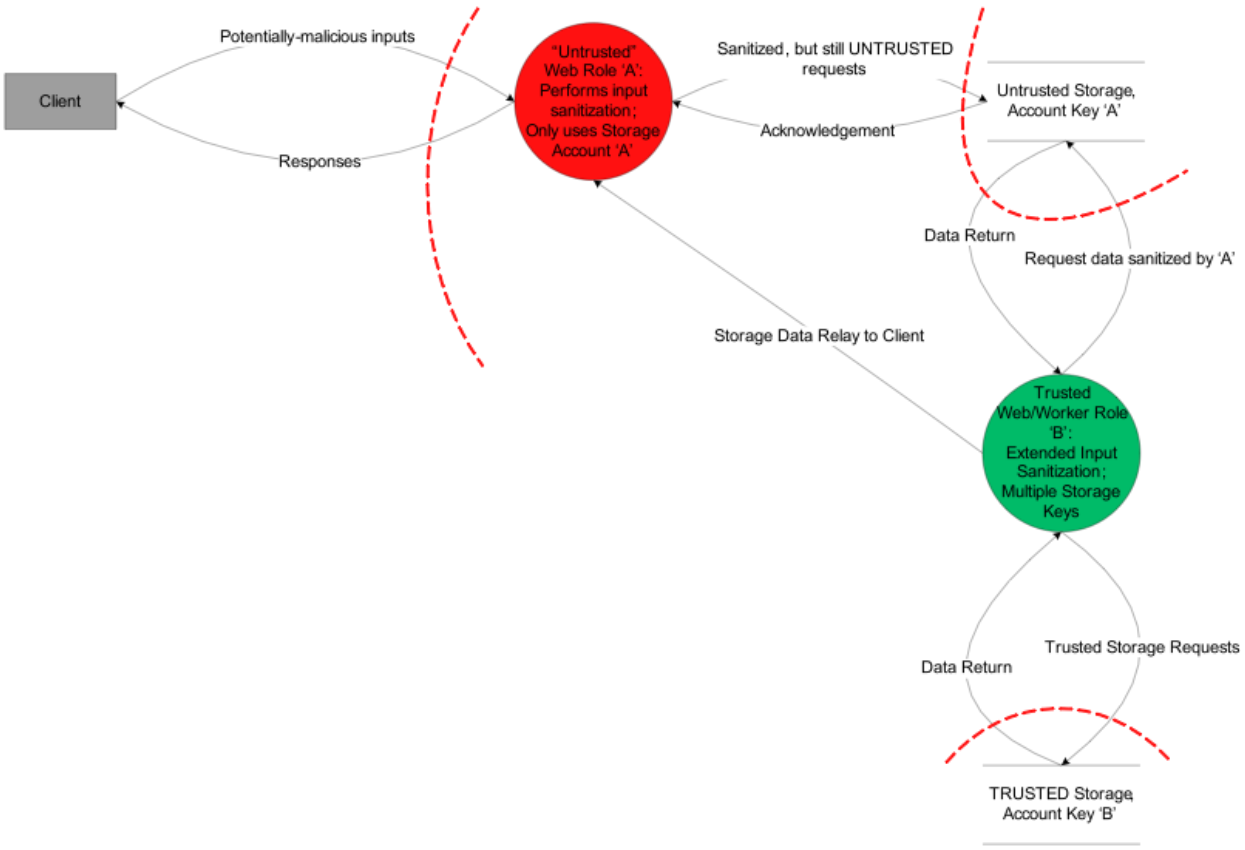

Figure 2: The multi-key design pattern

The multi-key design pattern has some advantages over the Gatekeeper/KeyMaster pattern:

- Providing separation of duty for storage accounts. In the event of Web Role A's compromise; only the untrusted storage account and associated key are lost.
- No internal service endpoints need to be specified. Multiple storage accounts are used instead.
- Windows Azure Partial Trust is not required for the externally-facing untrusted web role. Since PHP does not support partial trust, the Gatekeeper configuration is not an option for PHP hosting.

Applying SDL Practices to Windows Azure Applications

This next section describes the security practices that should be considered when designing and building Windows Azure applications.

The Microsoft Security Development Lifecycle (SDL) applies equally to applications built on the Windows Azure platform and any other platform. Most Windows Azure applications have been built, or will be built using agile methods. As a result, the SDL for agile process may be more applicable to applications hosted on Windows Azure than to the classic phase-based SDL.

The Microsoft SDL Web site covers SDL for Agile in detail.

Microsoft requires that the SDL be followed for any Microsoft-developed software deployed on Windows Azure. The SDL addresses security threats throughout the development process by means that include threat modeling during the design process; following development best practices and code security standards during coding; and requiring various tools for testing and verification before deployment. These proactive checks during development make software less vulnerable to potential threats after release, and the SDL provides a structured and consistent methodology with which to apply them. These methodologies, which are supported by an executive commitment to security, have helped Microsoft develop more secure software. Anyone who develops software in Windows Azure can use these same methods to improve security.

Security Education and Awareness

If a development team does not understand the basics of secure design and development, or the risks of running web-based software and services, then security training is imperative, and it should be completed before any Windows Azure application is designed, built, tested or deployed. All members of software development teams should be informed about security basics and recent trends in security and privacy, and they should attend at least one relevant security training class every year - at a minimum. Development team members should be encouraged to seek opportunities for additional security and privacy education. Developers who are well-versed and up-to-date on security issues are better able to design and develop software with security in mind first and foremost -- and not as an afterthought or "bolt-on" feature added at the end of the development process.

Given that Windows Azure applications are usually web-based managed code (ASP.NET) applications, appropriate topics for security education include:

- **Secure design**, including the following topics:
  - Attack surface reduction
  - Defense in depth
  - Principle of least privilege
  - Threat modeling
- **Secure coding**, including the following topics:
  - Cross-site scripting
  - SQL injection
  - Managed code security (transparency, code access security, assembly strong naming, etc.)

The SDL Process Guidance documentation provides links to books and training materials that are useful for beginning an SDL training program.

Secure Development Practices on the Windows Azure platform

The SDL provides guidance for use of development tools and practices that are applicable on the Azure platform. For the current version of compilers, linkers and other tools mandated by the SDL, see the SDL Process Appendix E.

- Use the SDL-required (or later) compiler versions to compile for the Win64 target platform:
  - C/C++ code: Visual C++ 2008 SP1
  - C# or Visual Basic .NET code: Visual C# 2005 and Visual Basic .NET 2005

- Compile and link native C/C++ code with /GS, /SAFESEH, /DYNAMICBASE and /NXCOMPAT. These options are enabled by default in Visual C++ 2008 SP1 and later. You can verify these settings using the BinScope binary analyzer.
- Native C and C++ code must not use banned versions of buffer handling functions. For more information, see Security Development Lifecycle (SDL) Banned Function Calls.
- Use the currently required (or later) versions of code analysis tools for native C and C++ (the /analyze compiler option).
- Run the FxCop code analysis tool against all managed code and fix all violations of the "Security" rules for the version of FxCop used.
- Follow data input validation and output encoding requirements to address potential cross-site scripting vulnerabilities.
- If using Microsoft SQL Azure database storage, only use parameterized queries or LINQ when accessing SQL-based data stores. Never dynamically construct a SQL statement from strings.
- Use an approved XML parser, such as .NET's System.Xml classes or, for native C++ code, use MSXML6 or XmlLite.
- Make sure your application is compliant with SDL cryptographic retirements laid out in "Appendix D: Using SDL-approved cryptography in Windows Azure" on page 26.

Verification and Release

Testing tools covered under SDL Process Guidance should be selected according to the specific technologies a Windows Azure application uses. Many of the tools used for COM/DCOM, RPC, or ActiveX testing, as well as for File Fuzzing, can be used on Windows Azure applications.

Conclusion

Computing solutions that use Windows Azure are very compelling to companies wishing to trim capital expenditure. However, security remains an important consideration. Software architects and developers must understand the threats to software developed for "the cloud" and use appropriate secure design and implementation practices to counter threats in the cloud environment. The progression from classic, client-server computing, to web-enabled applications, to applications hosted in the cloud, has changed the boundaries of applications, and these boundary shifts make understanding the threats to Windows Azure-based software all the more important. The work required to develop secure Windows Azure applications isn't new, revolutionary, or technically challenging; it simply requires that designers and developers consider the potential threats to their applications and apply the practices described in this paper.

Additional Resources

Microsoft Security Development Lifecycle (SDL) Portal
http://www.microsoft.com/sdl

Windows Azure Portal
http://www.microsoft.com/windowsazure/windowsazure/

Windows Azure Developer Portal
http://dev.windowsazure.com

Identity Developer Training Course
http://channel9.msdn.com/learn/courses/IdentityTrainingCourse/

Windows Identity Foundation
http://msdn.microsoft.com/en-us/security/aa570351.aspx

Active Directory Federation Services
http://www.microsoft.com/windowsserver2008/en/us/ad-fs.aspx

Microsoft's Compliance Framework for Online Services
http://www.globalfoundationservices.com/documents/MicrosoftComplianceFramework1009.pdf

Securing Microsoft's Cloud Infrastructure
http://www.globalfoundationservices.com/security/index.html

About Cross-Frame Scripting and Security
http://msdn.microsoft.com/en-us/library/ms533028(VS.85).aspx

Appendix A. Glossary

Developer Portal	The Windows Azure Developer Portal is an administrative portal for managing, deploying and monitoring Windows Azure services. The Developer Portal can be accessed at http://windows.azure.com
Fabric	The logical clusters of machines which provide a role execution environment inside a virtual machine.
Partial Trust	Partial trust is a concept in .NET that allows executable code to run with reduced capabilities such as the ability to print, make a socket connection or open files.
REST	REpresentational State Transfer; a software design that uses a stateless client-server architecture in which the web services are viewed as resources and can be identified by their URLs.
SAML	Security Assertion Markup Language. An XML-based industry standard for exchanging authentication and authorization information.
SDL	The Microsoft SDL is a security assurance process that is focused on software development. It is a collection of mandatory security activities, grouped by the phases of a software development life cycle (SDLC). You can learn more about the Microsoft SDL at http://www.microsoft.com/security/sdl/default.aspx
Svchost	Svchost is a process for hosting Windows services. The services are implemented as DLLs.
VM	A software emulation of a computer that runs in an isolated partition of a real computer.
VMBus	The subsystem that transfers data between the root operating system and guest virtual machines
Web Role	A web role is a role that is customized for web application programming as supported by IIS 7 and ASP.NET.
Worker Role	A worker role is a role that is useful for generalized development and may perform background processing for a web role

Appendix B. Windows Azure Deployment Security Threat Matrix

This appendix focuses on security issues specific to customer-deployed services that run on Windows Azure, to help highlight areas of increased risk and complexity compared to Software as a Service (SaaS) deployments using traditional web services. The goal of this appendix is to help Windows Azure developers and customers understand which security threats are mitigated by the Windows Azure environment and which security threats must be mitigated by the developer.

This appendix does not enumerate every conceivable threat to an application running on the Windows Azure infrastructure, nor does it address regulatory compliance issues.

Legend:

Mitigation is transparent to the customer, no action required.

Mitigation is provided by underlying platform/infrastructure and must be utilized by the customer's web application/service; Requires calls to existing/provided APIs.
Mitigation is per-service and is not provided at a lower level; Must be mitigated by the customer's code.
Mitigation is not yet implemented (planned for a future version of Windows Azure) but will be transparent to the customer once complete.
Mitigation is not yet implemented (planned for a future version of Windows Azure) and will require customer code to use Windows Azure APIs once complete.

Threat	**Layer where mitigation is implemented**	**Nature of mitigation provided (if specific to Windows Azure)**	**Application/Service-layer mitigation required**	**Is this issue higher risk or more complex in cloud deployments?**
Spoofing				
ARP Flooding	Platform	VM Switch	None Required	No
IP address spoofing	Infrastructure & Platform	Top-of-rack switches restrict which IP and MAC addresses VMs use	None Required	No
DNS spoofing	Infrastructure	Microsoft Live DNS Services	None Required	No
Tampering				

Packet tampering/interception on VM Bus	Platform	Trusted Channel between Hypervisor and VM tenants, VM switch has additional packet filters imposed	None Required	Yes
Windows Azure OS binary tampering	Platform	Binaries are Microsoft-signed and managed assemblies are strong named	Verify the signature of Windows Azure SDK binaries referenced in the application code.	Yes
Local Filesystem/Registry Tampering by compromised web services	Platform	Web Roles run as non-admin; strong ACLs on file system/registry enforced by runtime	None Required	Yes
Tampering/disclosure of credentials or other sensitive application data	Web Role		Use Windows Identity Foundation and HTTPS mutual authentication for SSL connections	No
Tampering with customer configuration data, encryption keys and intellectual property during web role provisioning	Infrastructure & Platform	VLANs, IP ACLs, Mutual SSL authentication in use between fabric controller and root/guest nodes.	None Required	No
Repudiation				
Audit log collection, storage and analysis	Platform and Web Role	Windows Azure Monitoring and Diagnostics APIs	Use monitoring and diagnostic APIs as needed; transfer logs to Storage private blob/table storage over HTTPS	Yes
Information Disclosure				
Footprinting or enumeration of services & applications in the VM	Platform	VLANs, HW & SW firewalls, VM switch filters, IP filtering in VM Guest	None Required	Yes

Side-channel attacks against VM Guests on the same physical host	Platform	1 VM per core, no communications between different tenants	None Required	Yes
Disclosure of data in transit between client and server	Platform and Web Role		Use HTTPS in place of HTTP where sensitive data is transferred	Yes
Disclosure of SSL Certificates/keys used by Web Roles	Platform and Web Role	Secure provisioning via Windows Azure Certificate Store	Use the certificate store for client and server SSL certificate storage	Yes
Disclosure of arbitrary secrets in blob/table/queue storage	Web Role/Client		Pre-encrypt secret data prior to uploading. Do not store decryption keys in Windows Azure Storage	Yes
Disclosure of Shared Access Signatures	Web Role/Client		Use HTTPS to securely transfer Shared Access Signatures to intended recipients and set appropriate permissions on containers.	Yes
Physical theft of storage account information, code or other intellectual property	Infrastructure	Physical Security and Operations Policies	None Required	No
Encrypted Storage of Arbitrary Secrets in Windows Azure Storage	Platform		Will require API calls similar to DPAPI	Yes
Denial of Service				
Denial of Service attacks via network bandwidth saturation (packet flooding)	Platform	Load balancing & throttling in network infrastructure	None Required	No

Identification of botnets and malicious network traffic	Infrastructure	Windows Azure Live Services monitors and investigates	None Required	Yes
Deep packet inspection for network attacks with known signatures	Platform		None Required	Yes
Flooding of Web Role local storage or blob/table storage	Platform	Quotas, ACLs, Reduced privilege execution and flood monitoring protection	None Required	Yes
Request flooding at the customer code/app level	Web Role		Implement application-level request throttling if necessary	No
Elevation of Privilege				
Anti-virus scanning of VM Guests/Hosts	Platform		None Required	Yes
Misconfiguration of Service/Application settings	Web Role		Must scope all cookies and the document.domain property to the service subdomain (eg. http://contoso.cloudapp.net) and NOT to *.cloudapp.net	Yes
Cross-site Request Forgery Attacks against the web role	Web Role		Use ASP.NET defenses	No
Cross-site Scripting Attacks against the web role	Web Role		Use the Anti-XSS Library	No
API fuzzing attacks on interfaces exposed by the web role	Web Role		Fuzz all interfaces and endpoints unique to code exposed to the web (or any other services)	No

Network packet fuzzing attacks against network protocols used by web role	Platform	IPFilter driver customizations fuzzed to SDL requirements, external penetration testing performed	None Required	No
File Fuzzing attacks against file parsers which are part of Windows Server 2008	Platform	Existing parsers fuzzed to SDL requirements, external penetration testing performed	None Required	No
File Fuzzing attacks against custom, application-provided file parsers	Web Role		Fuzz test all proprietary network protocol or file format parsers	No
Patching of security vulnerabilities at the Web Role/customer code level	Web Role		Have a security response and updating plan in place	No
API Fuzzing attacks against the Hypervisor root	Platform	APIs fuzzed to SDL requirements, external penetration testing performed	None Required	Yes

Appendix C: Excerpt from the Windows Azure Partial Trust Policy Reference

This is an abbreviated version of the full Windows Azure Partial Trust Policy available at the MSDN Portal. By default, roles deployed to Windows Azure run under full trust. To run your role under partial trust, you must add the enableNativeCodeExecution attribute on the WebRole or WorkerRole element and set it to false.

Legend:

WA Partial Trust is more restrictive here than ASP.Net Medium Trust
WA Partial Trust behavior differs from ASP.Net Medium Trust without being more restrictive
WA Partial Trust behavior is the same as ASP.Net Medium Trust

Permission	State	ASP.NET medium trust	Windows Azure partial trust
AspNetHosting	Level	Medium	Medium
EnvironmentPermission	Unrestricted	TEMP; TMP; USERNAME; OS; COMPUTERNAME	TEMP;TMP
	Read	TEMP; TMP; USERNAME; OS; COMPUTERNAME	TEMP;TMP
	Write	TEMP; TMP; USERNAME; OS; COMPUTERNAME	TEMP;TMP
FileIOPermission	Unrestricted	Denied	Denied
	Read	$AppDir$	$AppDir$; Any named local store
	Write	$AppDir$	Any named local store
	Append	$AppDir$	Any named local store
	PathDiscovery	$AppDir$	$AppDir$; Any named local store
IsolatedStorageFilePermission	Unrestricted	Denied	Denied
	AssemblyIsolationByUser	Permitted	Denied
	UnrestrictedUserQuota	Permitted	Denied
RegistryPermission	Unrestricted	Denied	Denied
SecurityPermission	Assertion	Permitted	Denied
	RemotingConfiguration	Permitted	Denied
SocketPermission	Connect	Denied	External sites only TCP
	Accept	Denied	Denied
SqlClientPermission	Unrestricted	Permitted	External sites only

WebPermission	Unrestricted	Denied	Denied
	Connect	$OriginHost$	External sites only
	Accept	Denied	Denied

Appendix D: Using SDL-approved cryptography in Windows Azure Applications

Many applications require the use of cryptographic technologies for encryption, tampering detection, signatures etc., and the SDL is prescriptive about which cryptographic technologies should be used. The following is a list of some of the more common requirements you will encounter when building a Windows Azure application.

- Where possible, use SSL/TLS (by using HTTPS) to transfer data between all parties, such as:

  Clients <-> Web and Worker Roles

  Clients <-> Storage endpoints

  Web Roles <-> Web and Worker Roles over internal endpoints

  Web and Worker Roles <-> Storage endpoints

- Web roles supporting HTTPS should be provisioned with certificates supporting 2048-bit RSA keys.
- Certificates should be renewed annually by refreshing the service configuration with the thumbprint of any new certificate uploaded via the developer portal.
- Use AES for symmetric cryptographic operations.
- Use 256-bit symmetric keys. This is especially important when a role needs to store encrypted data in Windows Azure Storage.
- Use RSA or Elliptic Curve Cryptography (ECC) for asymmetric cryptographic operations.
- Use RSA keys that are 2048-bit or longer.
- Use a SHA-2 algorithm (SHA-256, SHA-384 or SHA-512) for hashing and message-authentication codes.
- Use the strong entropy provisioned into the VM if keys are derived in role code. Windows Azure virtual machines are specially provisioned with strong entropy at boot.
- For more guidance on use of cryptography, see http://msdn.microsoft.com/en-us/security/sdl-process-guidance.aspx
- Store SSL/TLS certificates with Windows Azure Certificate Services, not in Windows Azure Storage or on local disk.