Global Foundation Services

Secure Use of Cloud Storage

BlackHat Briefings 2010

Grant Bugher
Lead Security Program Manager, Online Services Security and Compliance
Global Foundation Services, Microsoft Corporation

Agenda

- Cloud Storage Systems
  - Microsoft Windows Azure platform
  - Amazon Web Services platform
- Database Attacks
- Database Attacks on Cloud-based Applications
- Defenses and Mitigations
  - Development Mitigations
  - Designing More Secure Applications

Audience and Objectives

- Audience: software designers, architects, and testers who wish to design, build, and deploy applications that leverage cloud storage
- Objectives: understand the security challenges and recommended approaches to design and develop more secure applications using cloud storage

Cloud Storage Systems

- Move the application database layer to the cloud
- Promise high performance, scalability, and availability
- Handle many types of data
  - Blobs
  - Tabular data
  - Queues
  - Relational databases

Cloud Storage Systems

- Microsoft Windows Azure Platform
  - Windows Azure Storage
    - Blob service
    - Table service
    - Queue service
  - SQL Azure
- Amazon Web Services
  - Simple Storage Service (S3)
  - SimpleDB
  - Relational Database Service

Windows Azure Storage: Blob Service

- Arbitrary binary objects (blobs)
- Designated by container name and blob name
- Can be very large and have associated metadata
- Containers can be public or private
  - Private containers require a Shared Access Signature to access
  - Blobs in public containers can be accessed with a browser
  - Public blobs are supported by Windows Azure Content Delivery Network
- Managed through REST API

Windows Azure Storage: Table Service

- Arbitrarily formatted tabular data
  - Each row has a series of named properties
  - Rows in the same table may have different properties
- Tables can be very large (billions of rows, terabytes of data)
- Not relational
- Managed and accessed through a REST API
  - Special headers required; not accessible in a browser
  - WCF Data Services API provided for .NET developers

Windows Azure Storage: Queue Service

- A queue of very short (4K) binary messages
  - Intended for communication between Windows Azure roles
  - Not intended for storing data at rest; messages expire
- Items can be masked or removed, up to 32 at a time
- Managed and accessed through a REST API
  - Special headers required; not accessible in a browser

SQL Azure

- True relational database service in the cloud
- Based on Microsoft SQL Server 2008
  - Supports tables, indexes, views, stored procedures, constraints, triggers
  - No service broker, SQL Reporting Services, SQL Analysis Services
- Accessed via Tabular Data Stream (TDS) protocol
  - Can use any Microsoft SQL Server client library
  - Managed via standard SQL Server tools

Amazon Web Services: S3

- Simple Storage Service (S3)
- Arbitrary binary objects (blobs)
- Designated by bucket name and object name
- Can be very large and have associated metadata
- Containers can be public or private
  - Private buckets require a Signed URL to access
  - Objects in public buckets can be accessed with a browser
  - Public objects are supported by Amazon CloudFront content delivery network

Amazon Web Services: SimpleDB

- Arbitrarily formatted tabular data
  - Each entity has a series of named attributes
  - Entities in the same domain may have different attributes
- Tables can be very large (billions of rows, terabytes of data)
- Not relational
- Managed and accessed through REST or SOAP APIs
  - REST API potentially accessible from a browser, but not intended
  - Java and .NET SDK binaries provided
  - WSDL provided for SOAP API to enable SOAP toolkits

Relational Database Service

- True relational database service in the cloud
- Based on MySQL
  - Full support for standard MySQL features
  - Accessed via MySQL Client/Server Protocol
  - Can use any MySQL client library
- SQL Azure and Amazon Relational Database Service are identical to standard, non-cloud database hosting

Injection Attacks

- Database layer is generally not exposed to the Internet
  - Not subject to direct attack
  - Must be attacked by way of the application layer
- Carefully crafted malicious input
  - Looks like valid data to the application
  - Executed as instructions by the database
- Injection attacks can be against SQL or XML

Structured Query Language (SQL)

- Structured Query Language
  - Used by Microsoft SQL Server, MySQL, Oracle – any RDBMS
  - Many variants: T-SQL, PL/SQL, etc.
- Contains multiple types of commands
  - Data Definition Language (DDL) defines schema
  - Data Manipulation Language (DML) performs data operations
- Sometimes supports additional functionality
  - MS SQL extended stored procedures & CLR stored procedures
  - Oracle PL/SQL procedural code

SQL Injection

- SQL queries are mixed code and data
- SQL query is fundamentally a string, containing:
  - verbs instructing the database what to do
  - identifiers of objects to be acted on
  - values to use in operations, often based on user data
- The difference between code and data is determined by syntax and delimeters

SQL Injection

```
SELECT 'id', 'username', 'permissions'
FROM 'users' AS 'u' INNER JOIN 'authorization' AS 'a'
WHERE 'u.username' == "grantb" AND 'u.id' == 'a.id';

SELECT 'id', 'username', 'permissions'
FROM 'users' AS 'u' INNER JOIN 'authorization' AS 'a'
WHERE 'u.username' == "x" OR 1=1; --" AND 'u.id' ==
'a.id';
```

SQL Injection

```
SELECT 'id', 'username', 'permissions'
FROM 'users' AS 'u' INNER JOIN 'authorization' AS 'a'
WHERE 'u.username' == "x" OR 1=1; INSERT INTO 'users'
VALUES 1000, "hacker"; INSERT INTO `permissions`
VALUES 1000, "admin"; --" AND 'u.id' == 'a.id';
```

```
SELECT 'id', 'username', 'permissions'
FROM 'users' AS 'u' INNER JOIN 'authorization' AS 'a'
WHERE 'u.username' == "x"; DROP TABLE 'users'; --"
AND 'u.id' == 'a.id';
```

SQL Injection Mitigations

- Input validation: Match user input against a whitelist when possible, and strip out known metacharacters such as quotation marks otherwise
- Encoding: Escape or transform metacharacters into a harmless form, such as URL encoding or base64
- Parameterization: Use data APIs such as those in .NET and Java to separate query code and syntax from potentially harmful data, and send them separately.

XML and XPATH Injection

- XML data and XPATH queries are also subject to injection
  - Same issue as with SQL: mixed code and data
  - XML contains both the data items and their schema
  - XPATH contains both the query structure and the actual items to be searched for
- XML data often uses smart serializers, which partially mitigate these effects automatically
- REST interfaces are so simple that developers often construct XML from strings

XML Parsers

- Simple API for XML (SAX)
  - Unidirectional event-driven stream parser
  - Reads XML serially, interpreting objects as it goes along
  - Fast, low-memory serial processing
  - No standards for behavior; Java parser is considered normative
- Document-Object Model (DOM)
  - Full in-memory representation of entire XML document
  - Allows random access to nodes
  - High memory requirements, but fast for random access
  - DOM Level 3 is a formal W3C standard

XML Injection Example

```
<user>
    <username>fred</username>
    <email>fred@contoso.com</email>
    <password>sdfyn2o49r</password>
    <uid>432</uid>
    <group>accounting</group>
    <hint>this is my password hint!</hint>
</user>
```

XML Injection vs. SAX parser

```
<user>
    <username>fred</username>
    <email>fred@contoso.com</email>
    <password>sdfyn2o49r</password>
    <uid>432</uid>
    <group>accounting</group>
    <hint></hint><uid>0</uid>
    <group>admin</group><hint>Hi!</hint>
</user>
```

XML Injection vs. DOM parser

```
<user>
     <username>fred</username></username>
     <email>fred@contoso.com<!--</email>
     <password>sdfyn2o49r</password>
     <uid>432</uid>
     <group>accounting</group>
     <hint>--></email><password>sdfyn2o49r</password>
<uid>0</uid><group>administrator</group><hint>this is
my password hint!</hint>
</user>
```

External Entity Injection vs. DOM parser

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE xy [
   <!ELEMENT xy ANY >
   <!ENTITY attack SYSTEM "file:///etc/passwd">]>
<xy>&attack;</xy>
```

Attacking Cloud Storage Applications

- Amazon Web Services and Microsoft Windows Azure use a variety of interfaces based on text strings
  - AWS SimpleDB query strings (SQL)
  - Azure Table Storage query strings (new query language)
  - AWS SOAP API (XML)
  - Azure Blob, Table, and Queue Storage APIs (XML)
- Due to the popularity of REST and SOAP for web services, future cloud data storage systems will likely do the same
- Text strings are potentially subject to injection

Attacking Cloud Storage Applications

- Variety of approaches
  - Query String Injection
  - Direct Node Injection
  - DOM injection via XML comments and !CDATA
  - Persistent cross-site scripting
  - Storage Enumeration
  - Account-level and Infrastructure Attacks
- All of these attack *applications* and *users* of cloud storage – they do not rely on any vulnerability in Azure or AWS

Query String Injection

- Amazon Web Services SimpleDB uses SQL-like queries
  - SELECT *attributes* FROM *domain* WHERE *conditions* ORDER BY *sort_instructions* LIMIT *count*
  - The entire query string is one parameter sent to the REST API
  - SimpleDB SDK still considers the entire string ("select expression") to be one parameter
- Azure Table Storage query string is not SQL-like
  - However, it is still a single parameter, encoded as a unit, and potentially subject to its own kind of injection attack

Query String Injection

- Since the query string is a single parameter, onus is on the application developer to prevent injection
- SimpleDB query strings support multiple quotation types (", ', `) and do not require escaping for one quotation type enclosed in another (similar to PHP behavior)
- An attacker who can control input to multiple fields can change the semantics of the query

Query String Injection

```
select * from products where category = 'mycategory'
intersection access = 'public' order by 'columnname'
desc

select * from products where category = 'mycategory'
or category = "' intersection access = 'public' order
by '" order by 'columnname' desc

select * from products where category = 'mycategory'
or category = "' intersection access = 'public' order
by '" order by 'columnname' desc
```

Query String Injection

Demo

Query String Injection

- Presence of multiple quotation types can allow users to insert or elide fields from the query
- Mitigation is identical to that for SQL Injection:
  - Validate user input
  - Encode values before constructing a query string
- No equivalent to parameterized queries
- The good news: query strings can only issue queries
  - No equivalent to DDL
  - No equivalent to CLR, PL/SQL, or stored procedures
  - Verb is found outside the query string

XML Injection

- Windows Azure Storage interfaces are XML sent over REST
- XML interfaces can also be subject to injection
  - Less familiar than SQL injection
  - Higher potential than query string injection, since write operations and DDL are possible
- Plan of Attack:
  - Send attacks for both types of parsers (SAX, DOM)
  - Do not make assumptions about SDK/API used

Direct Node Injection

```
<content type="application/xml">
    <m:properties>
      <d:Title>This is my comment title</d:Title>
      <d:UserID>523</d:UserID>
      <d:AvatarUrl>http://myaccount.blob.core.windows.net/guestbookpics/
image_65fa49ae-1a73-4882-a15b-1b46389b855d.jpg</d:AvatarUrl>
      <d:CommentText>This is my comment's contents</d:CommentText>
      <d:PartitionKey>06232010</d:PartitionKey>
      <d:RowKey>12521249962450735715_5c2dbee9-66d4-4e51-92ce-
b8100055b635</d:RowKey>
      <d:Timestamp m:type="Edm.DateTime">2010-07-01T12:22:01</d:Timestamp>
    </m:properties>
  </content>
```

Direct Node Injection

```
<content type="application/xml">
    <m:properties>
      <d:Title>This is my comment title</d:Title>
      <d:UserID>523</d:UserID>
      <d:AvatarUrl>http://myaccount.blob.core.windows.net/guestbookpics/
image_65fa49ae-1a73-4882-a15b-1b46389b855d.jpg</d:AvatarUrl>
      <d:CommentText></d:CommentText><d:UserID>100</d:UserID><d:CommentText>
This is a message.</d:CommentText>
      <d:PartitionKey>06232010</d:PartitionKey>
      <d:RowKey>12521249962450735715_5c2dbee9-66d4-4e51-92ce-
b8100055b635</d:RowKey>
      <d:Timestamp m:type="Edm.DateTime">2010-07-01T12:22:01</d:Timestamp>
    </m:properties>
  </content>
```

Direct Node Injection

- …but it doesn't work.
  - Platform protects from this sort of attack
  - Doesn't work even if you send it to the data endpoint directly
- Duplicate nodes invalidate the entire API call
  - 400 Bad Request
  - No other error information returned, just ambiguous failure
- Almost certainly a DOM parser at work

XML Injection with Comments and CDATA

- We know the parser is smart – a DOM parser
- Smart parsers usually implement the full XML specification
  - XML comments with <!-- … -->
  - Block encoding with <![CDATA[…]]>
- Injecting these tags may allow modifying the XML
  - The meaning is changed yet no duplicate tags are introduced; the XML parses without errors
  - Unfortunately for attackers, this introduces "junk" into one or more tags

XML Comment Injection - Azure

```
<content type="application/xml">
    <m:properties>
      <d:Title>This is a message<!--</d:Title>
      <d:UserID>523</d:UserID>
      <d:AvatarUrl>http://myaccount.blob.core.windows.net/guestbookpics/
image_65fa49ae-1a73-4882-a15b-1b46389b855d.jpg</d:AvatarUrl>
      <d:CommentText>--></d:Title><d:UserID>200</d:UserID>
<d:AvatarUrl>http://myaccount.blob.core.windows.net/guestbookpics/
image_65fa49ae-1a73-4882-a15b-1b46389b855d.jpg</d:AvatarUrl><d:CommentText>This
is a comment</d:CommentText>
      <d:PartitionKey>06232010</d:PartitionKey>
      <d:RowKey>12521249962450735715_5c2dbee9-66d4-4e51-92ce-
b8100055b635</d:RowKey>
      <d:Timestamp m:type="Edm.DateTime">2010-07-01T12:22:01</d:Timestamp>
    </m:properties></content>
```

XML CDATA Injection - Azure

```
<content type="application/xml">
    <m:properties>
      <d:Title>This is a message<![CDATA[</d:Title>
      <d:UserID>523</d:UserID>
      <d:AvatarUrl>http://myaccount.blob.core.windows.net/guestbookpics/
image_65fa49ae-1a73-4882-a15b-1b46389b855d.jpg</d:AvatarUrl>
      <d:CommentText>]]></d:Title><d:UserID>200</d:UserID>
<d:AvatarUrl>http://myaccount.blob.core.windows.net/guestbookpics/
image_65fa49ae-1a73-4882-a15b-1b46389b855d.jpg</d:AvatarUrl><d:CommentText>This
is a comment</d:CommentText>
      <d:PartitionKey>06232010</d:PartitionKey>
      <d:RowKey>12521249962450735715_5c2dbee9-66d4-4e51-92ce-
b8100055b635</d:RowKey>
      <d:Timestamp m:type="Edm.DateTime">2010-07-01T12:22:01</d:Timestamp>
    </m:properties></content>
```

XML Comment Injection - SimpleDB

```
<soapenv:Body>
<PutAttributesRequest xmlns='http://sdb.amazonaws.com/doc/2009-04-15'>
      <Attribute><Name>Title</Name><Value>This is a message<!--</Value></Attribute>
      <Attribute><Name>UserID</Name><Value>523</Value></Attribute>
      <Attribute><Name>AvatarURL</Name><Value>http://myaccount.blob.core.windows.net/
guestbookpics/image_65fa49ae-1a73-4882-a15b-1b46389b855d.jpg</Value></Attribute>
<Attribute><Name>CommentText</Name><Value>--><Name>UserID</Name><Value>200</Value>
</Attribute><Attribute><Name>AvatarURL</Name><Value>http://myaccount.blob.core.windows
.net/guestbookpics/image_65fa49ae-1a73-4882-a15b-1b46389b855d.jpg</Value></Attribute>
<Name>CommentText</Name><Value>This is a comment</Value></Attribute>
      <DomainName>MyDomain</DomainName>
      <ItemName>Message012354</ItemName>
      <Version>2009-04-15</Version>
    </PutAttributesRequest>
</soapenv:Body>
```

Persistent Cross-Site Scripting

- Encoding input is often used to prevent XML attacks
  - `<!--` becomes `&lt;!--`
  - `<![CDATA[` becomes `&lt;![CDATA[`
- Encoding output is also used to prevent cross-site scripting
  - `<script>alert(‘xss’);</script>` becomes `&lt;script&gt;alert(‘xss’);&lt;/script&gt;`
- These encodings do not substitute for each other!
- Input encoding on data storage is reversed before output

XML Injection

Demo

Persistent Cross-Site Scripting

```
<content type="application/xml">
    <m:properties>
      <d:Title>&lt;script&gt;alert(&apos;xss&apos;);&lt;/script&gt;</d:Title>
      <d:UserID>523</d:UserID>
      <d:AvatarUrl>http://myaccount.blob.core.windows.net/guestbookpics/
image_65fa49ae-1a73-4882-a15b-1b46389b855d.jpg</d:AvatarUrl>
<d:CommentText>&lt;script&gt;alert(&apos;xss&apos;);&lt;/script&gt;</d:CommentT
ext>
      <d:PartitionKey>06232010</d:PartitionKey>
      <d:RowKey>1252124996245073571 5_5c2dbee9-66d4-4e51-92ce-
b8100055b635</d:RowKey>
      <d:Timestamp m:type="Edm.DateTime">2010-07-01T12:22:01</d:Timestamp>
    </m:properties>
  </content>
```

Enumeration

- Cloud storage is often used to store public or semi-public files for end-user download
  - Massive storage capacity
  - Cheap, scalable bandwidth
- Subscription-based sites run into difficulties
  - Files not hosted behind site access control
  - Often resort to security by obscurity
  - Difficulty with tiered access levels

S3 Ripper

Demo

Preventing Enumeration

- Use of public buckets in access-controlled or multitenancy situations is inappropriate
  - Container/bucket names may be exposed on public forums
  - Members of one tenant or access control level may access others
- Ineffective Mitigations
  - Flash downloader on a site with referrer-checking
  - Registered a separate domain CNAMEd to the AWS bucket
  - Provide a script on an access-controlled site that proxies the download

Preventing Enumeration

- Container-level access control
  - Windows Azure has a setting to make blobs public but not allows container access
    - `x-ms-blob-public-access=blob`
  - Prevents enumeration, but not URL sharing of files
  - No enumeration tools for Azure Storage are currently available
- The right solution: private containers
  - Windows Azure Storage Shared Access Signatures
  - Amazon S3 Signed URLs

Account-Level Attacks

- In addition to writing secure applications, the storage account itself must be protected
- Both AWS and Windows Azure use an email address & password as an administrative credential
  - Never use the admin's normal Amazon account or Live ID
  - Should be treated like a Domain Admin or root password
  - Ensure complete control of the email account's domain
    - Webmail providers like Hotmail or GMail often have password reset
    - Entire enterprises have been compromised via email password reset
- Use multifactor authentication where available

Defenses and Mitigations

- Input Validation
- Encoding
- Windows Communication Foundation
- SOAP toolkits and WSDL-based code generation
- Encryption
- Multifactor authentication
- Shared access signatures and Signed URLs
- Application design and defense in depth

Input Validation

- All of these attacks come from malicious user input
- Blacklist-based platform mitigations
  - ASP.NET ValidateRequest
  - Apache mod_security
- Whitelist-based mitigations
  - Allow only the types of data you expect
  - Easy for formatted types like ZIP, SSN, or SHA-256 hash
  - Very difficult for XML/HTML input or Unicode global applications
- Microsoft Web Protection Library AntiXSS Encoders (http://wpl.codeplex.com/)

Encoding

- Render potentially malicious input or output harmless
- XML Encoding vs. data storage attacks
  - Can wrap user input in a <![CDATA[ ]]> block, but watch for closing sequences in the input
  - HTML/XML encoding is fairly standard with few metacharacters, and many encoding libraries available (including AntiXSS)
- HTML Encoding vs. cross-site scripting
  - Encode output before sending it to the user
  - Separate from data storage encoding
  - Still not foolproof – multiple encoding attacks are very sophisticated especially with multi-tier globalized applications

Windows Communication Foundation

- .NET libraries for abstract data access
  - Application uses object model or LINQ queries
  - WCF translates these to appropriate protocol based on user code
  - Can be used with any storage back-end
- Default interface for Azure Table Storage in ASP.NET
  - StorageClient assembly provided in Windows Azure SDK
  - Automatically performs proper XML encoding
  - Roughly equivalent to parameterizing queries
- No vulnerabilities found in WCF encoders
  - Only works if you use it – not every application is in ASP.NET
  - Developer still constructs query strings

SOAP Toolkits

- SOAP is rarely written by string concatenation
  - Most AWS developers use REST API
  - SOAP API interfaces are generated from WSDL with tools
- Many SOAP toolkits are available
  - System.Web.Services namespace & WSDL tool in Visual Studio .NET
  - Similar tools available for Java, PHP, and other platforms
- Generated interface code generally performs necessary XML encoding and CDATA usage

Encryption

- Use HTTPS when communicating with storage endpoints
  - AWS forces you to anyway – no HTTP option in many cases
  - Very little reason not to – you don't pay for compute time on storage nodes in Azure or AWS
- Critical if communicating over the Internet
- Should still use HTTPS even when communicating inside the cloud service provider's services
  - VMs are shared with other customers
  - Provider datacenter is a "black box" to you
  - Probably very safe, but very little reason not to do it

Multifactor Authentication

- Amazon Web Services
  - Offers administrator multifactor authentication via a dynamic password token
  - Nominal fee to purchase token
  - One level of access
- Microsoft Windows Azure
  - Developer access can be via certificates, including smart card certificates at no charge
  - Supported by development tools like Visual Studio .NET
  - One level of access
  - Administrative login with password cannot be disabled

Shared Access Signatures

- Prevent enumeration and unauthorized access to blobs/objects in cloud storage
- Shared Access Signatures (Azure)/Signed URLs (AWS)
  - Set blob or object as private
  - Application issues signed URLs, accessible in the browser, that are valid only for a limited time and only for the specified object
- User is not provided direct links to files
  - Link to an application page which generates a signed URL for the desired object and only for a short time, then redirects browser
  - Transparent to users, but makes public sharing of URLs difficult and short-duration

Windows Azure Shared Access Signatures

```
http://myaccount.blob.core.windows.net/musicdownloads
/filename.mp3

http://myaccount.blob.core.windows.net/musicdownloads
/filename.mp3?st=2010-07-28T11:20Z&se=2010-07-
29T12:20Z&sr=c&sp=r&si=YWJjZGVmZw%3d%3d&sig=dD80ihBh5
jfNpymO5Hg1IdiJIEvHcJpCMiCMnN%2fRnbI%3d
```

Application Design Mitigations

- Defense in Depth
  - Use multiple layers of defense
  - Keep compromises isolated
  - Prevent elevation of privilege and cascading compromise
- Consider platform defenses
  - Layers of protection
  - Use cloud equivalents of traditional defenses
- Design the application for security
  - Gatekeeper design pattern

Service-Layer Security Considerations

Service Layer

- **Your product's security architecture, compliance controls, and code quality**
- **Use of platform-provided threat mitigations and security features**
- **Mitigation of traditional web application security threats**

Platform Layer

- **Windows Azure runtime privileges and CAS trust levels**
- **User accounts, isolation, and operation system privilege levels**
- **Secure Interaction with Platform Services for authentication, authorization, audit, and storage**

Infrastructure Layer

- **Service scale-out size and duration**

Mapping Traditional Defenses to the Cloud

Input Validation, Sanitization, Fuzzing	No change
Scoping Issues, Application-level DoS protection	Subdomain scope, Application request throttling
Authentication, Authorization, Audit	ADFSv2, WLID, ACS, MDS
Storage ACLs	Shared-Access Signatures
Certificate Services	WACS via Azure Development Portal
IPC	Internal Endpoints

Defenses Inherited by Azure Tenants

Spoofing

- VLANs
- Top of Rack Switches
- Custom packet filtering

Repudiation

- Monitoring / Diagnostics Service

Tampering & Disclosure

- VM switch hardening
- Certificate Services
- Shared-Access Signatures
- HTTPS
- Sidechannel protections

Denial of Service

- Load-balanced Infrastructure
- Network bandwidth throttling
- DDoS protection on Storage nodes
- Configurable scale-out

Elevation of Privilege

- Partial Trust Runtime
- Hypervisor custom sandboxing
- Virtual Service Accounts

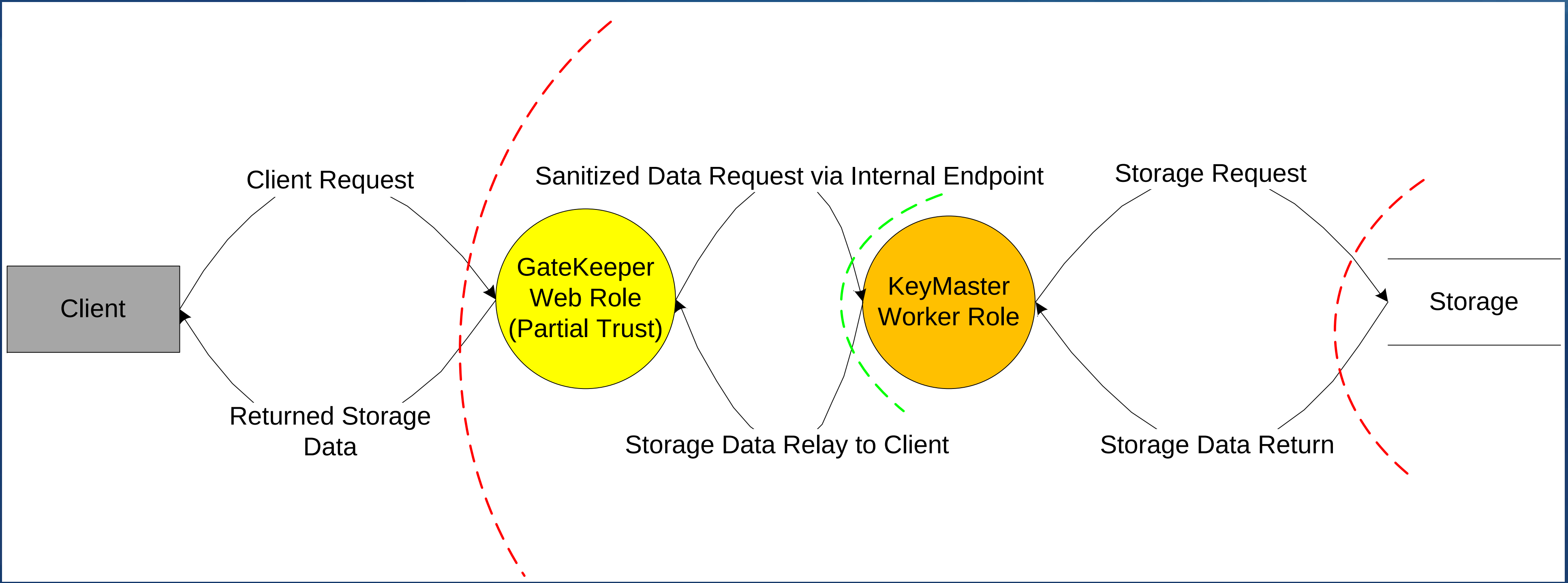
Developing a Secure Cloud Application

- The Microsoft Security Development Lifecycle (SDL) is applicable to Windows Cloud applications
  - SDL for Agile likely to be used more often
  - www.microsoft.com/sdl covers SDL for Agile in detail
  - Mandatory for Microsoft-developed software deployed in Azure
  - Addresses security threats throughout the development process
    - Threat modeling, best practices, test tools
    - Performing checks proactively during development reduces bugs at release
- These methods can be used by anyone to develop more secure applications

Developing a Secure Azure Application

- Appropriate design patterns can take advantage of Azure runtime trust models to create hardened applications
- Isolate web roles and separate duties of individual roles to maximize the use of Windows Azure Partial Trust
  - Secure externally-facing web roles
  - Restricts access to file system, registry, local environment, sockets, and web connections
- Gatekeeper design pattern can separate privileged and unprivileged roles

Gatekeeper Design Pattern


Client
Client Request
Returned Storage Data
GateKeeper Web Role (Partial Trust)
Sanitized Data Request via Internal Endpoint
Storage Data Relay to Client
KeyMaster Worker Role
Storage Request
Storage Data Return
Storage

Gatekeeper Design Pattern

- Gatekeeper role
  - Services requests from the Internet
  - Windows Azure Partial Trust
- Keymaster role
  - Privileged back-end worker role
  - Takes inputs only via a secure channel from the Gatekeeper
  - Handles sanitized storage requests
  - May be run in Windows Azure Full Trust with Native Code Execution if necessary for the application

Gatekeeper Pattern Outside Azure

- Windows Azure trust model facilitates this design pattern
  - ...but you can implement it anywhere, including AWS
  - Separate presentation logic from storage logic
  - Create a restricted boundary between tiers
- Configure least privilege on each tier
- Threat model back-end tiers under the assumption that the front-end tiers are already compromised

Conclusion

- Security an important consideration in the cloud
  - The work required to secure an application using cloud storage is not new or more technically challenging than a traditional database environment
  - Designers and developers must consider the threats to their application and follow appropriate practices
- Must use appropriate secure design and implementation patterns to counter threats in a cloud environment

Microsoft®

Your potential. Our passion.™