# CloudFilter: Practical Control of Sensitive Data Propagation to the Cloud

Ioannis Papagiannis
Imperial College London
London, SW7 2AZ
United Kingdom
ip108@doc.ic.ac.uk

Peter Pietzuch
Imperial College London
London, SW7 2AZ
United Kingdom
prp@doc.ic.ac.uk

## ABSTRACT

A major obstacle for the adoption of cloud services in enterprises is the potential loss of control over sensitive data. Companies often have to safeguard a subset of their data because it is crucial to their business or they are required to do so by law. In contrast, cloud service providers handle enterprise data without providing guarantees and may put confidentiality at risk. In order to maintain control over their sensitive data, companies typically block all access to a wide range of cloud services at the network level. Such restrictions significantly reduce employee productivity while offering limited practical protection in the presence of malicious employees.

In this paper, we suggest a practical mechanism to ensure that an enterprise maintains control of its sensitive data while employees are allowed to use cloud services. We observe that most cloud services use HTTP as a transport protocol. Since HTTP offers well-defined methods to transfer files, inspecting HTTP messages allows the propagation of data between the enterprise and cloud services to be monitored independently of the implementation of specific cloud services. Our system, CLOUDFILTER, intercepts file transfers to cloud services, performs logging and enforces data propagation policies. CLOUDFILTER controls where files propagate after they have been uploaded to the cloud and ensures that only authorised users may gain access. We show that CLOUDFILTER can be applied to control data propagation to Dropbox and GSS, describing the realistic data propagation policies that it can enforce.

## Categories and Subject Descriptors

H.3.2 [**Information Storage and Retrieval**]: Information Storage; D.4.6 [**Operating Systems**]: Security and Protection; C.2.0 [**Computer-Communication Networks**]: General

## General Terms

Security

## Keywords

Information Flow Control, Cloud Storage, Data Loss Prevention


*CCSW'12,* October 19, 2012, Raleigh, North Carolina, USA.


## 1. INTRODUCTION

The potential loss of control over sensitive data is a major obstacle for the adoption of cloud services in enterprises. Cloud services typically do not provide strong guarantees regarding data confidentiality. For example, Dropbox [4], a popular cloud storage service, was shown to suffer from vulnerabilities that could disclose files in various scenarios [15]. Enterprises, especially when legally liable for data loss, are thus reluctant to adopt cloud services [23].

A common practice to maintain control over sensitive enterprise data is to prohibit all communication with specific services at the network level. This has two shortcomings: first, employees cannot execute workflows involving cloud services with data that are not sensitive. For example, blocking access to Dropbox would prevent sales staff from collecting client requirements using a shared document; second, blocking popular services makes it more likely for users to bypass such a security policy completely. For example, an ambitious salesperson may choose to rely on their personal laptop and thus expose Dropbox to arbitrary company documents.

In general, enterprises impose different restrictions on different data with regard to cloud usage: some data are public and no restrictions exist; others are sensitive and should never leave the enterprise network. Some categories of data may be stored in the cloud as long as the enterprise has reasonable guarantees that only specific users may have access, i.e. a specific *data propagation policy* is enforced. Moreover, and in contrast to military systems, the decision to classify data in a particular category is often made by employees, which assumes that employees are accountable for their actions. Similar requirements may also be introduced by service providers, e.g. a provider may choose not to host particular categories of data if it cannot provide adequate protection.

Existing access control schemes are insufficient to prevent data disclosure when cloud services are used. Information Rights Management systems [18] expect employees to release only encrypted documents but there is no guarantee that employees comply with such a policy. Information Flow Control systems [26] can monitor each employee's actions transparently but are impractical to deploy because they require that both employee machines and the cloud provider's systems are modified to propagate security labels.

Our work is based on the observation that, for most cloud services, data is transfered between clients and services using HTTP. In other words, HTTP replaces IP as the neck in the traditional hourglass-shape model of the network protocol stack [2]. When controlling data propagation, HTTP thus offers well-defined methods to transfer files [10], which are independent of a provider's API and are easy to intercept.

We propose CLOUDFILTER, a practical and service-independent system to restrict how data propagates between the enterprise and the cloud provider. We exploit two ideas to enforce data propaga-

tion policies effectively. First, we bind data propagation policies to the files controlled by these policies using *security labels embedded inside files*. Since embedded labels remain associated with files, we can decide how a particular file should propagate when it is first uploaded to a cloud storage service and then enforce this decision in any subsequent attempt to download the same file. No additional support is required from the cloud provider for this. Second, we modify the employee's browser to expose *contextual information* such as the name of the user that performs each upload. Again, this avoids the need for solutions specific to particular cloud providers.

The architecture of CLOUDFILTER consists of proxies that intercept HTTP traffic at the perimeter of the enterprise and the cloud provider's networks and enforce data propagation policy. When a file transfer is detected, proxies pause it, may ask the user for confirmation and log it for future audit. A browser extension lets users choose propagation policies for their files. Proxies attach security labels to files in transit. Labels store all the information required to restrict file propagation, e.g. the domain that the document originates from or a list of users eligible to receive it.

CLOUDFILTER can ensure that, when an enterprise user uploads a document to a cloud storage service, (1) the operation is logged, (2) the action can be attributed to a user and (3) other users can only access this document from a designated set of networks. This makes an enterprise more likely to adopt a cloud service—at least for a subset of its data—because monitoring, accountability and policy-imposed restrictions on data propagation reduce the fear of uncontrolled, accidental and large-scale data disclosure.

The paper is structured as follows. The next section introduces the threat model that CLOUDFILTER is designed to protect against and discusses existing solutions for controlling data propagation. §3 presents an overview of CLOUDFILTER and describes the specification of data propagation policies. §4 demonstrates use cases for CLOUDFILTER with two representative cloud storage providers and discusses limitations. The paper closes in §5 with future work.

# 2. BACKGROUND

Maintaining control over data is an important concern for organisations considering migration to the cloud, particularly in government and commerce [23]. Cloud providers offer services that are easy to access and less expensive than in-house equivalents but they seldom offer strong guarantees about data confidentiality. This creates a hybrid policy in many enterprises: the propagation of a subset of enterprise data must be strictly controlled while the cloud should only be used for files that are not sensitive.

## 2.1 Threat Model

We aim to control the propagation of data for cases, in which users are not malicious. The scenario that we protect against is of a user who inadvertently uploads sensitive data to the cloud and then shares or accesses the data in a way that violates company policy. Users are considered trusted to decide whether a document falls under a particular security category and how a document should be shared with other users. At the same time, employees can be held accountable for their actions. Scenarios in which a user actively tries to circumvent the protection mechanism can be handled using disciplinary actions.

We believe that these assumptions capture the trust that most enterprises have in their employees. Employees can typically send arbitrary documents to any recipient, e.g. via company email. At the same time, an enterprise wants assurance that certain policies are enforced (e.g. no large messages can be sent) and that any violations can be attributed to a particular employee for disciplinary actions (i.e. by keeping email logs).

For data propagation policies that require the collaboration of the cloud service, we assume that the enterprise trusts the cloud provider to correctly deploy and configure a local installation of CLOUDFILTER. We anticipate that cloud providers have an incentive to collaborate with enterprises: allowing enterprises to restrict the propagation of their data can give cloud providers a distinguishing feature in the marketplace. It also means that they cannot be held liable after confidential data was compromised.

Many solutions have been proposed to control the propagation of sensitive data. Representative schemes employ Information Rights Management (IRM) and Information Flow Control (IFC). In addition, Data Loss Prevention (DLP) systems target specifically confidential data disclosure in enterprises.

## 2.2 Information Rights Management

IRM systems [18, 12] (also known as Digital Rights Management systems) control access to sensitive enterprise documents. Typically, a server stores an unencrypted version of the document along with access control lists. When a document is retrieved from the server, it is encrypted for distribution. Recipients of the document need to access the server, authenticate and obtain a decryption key before displaying it.

IRM systems are impractical when used with cloud services. They normally require customised client software (e.g. document editors) and access to a shared trusted authentication server. While this is a reasonable assumption within a single enterprise, it is seldom the case when two employees in different enterprises collaborate on a shared document. Second, there is no guarantee that employees use the IRM system before uploading a document to the cloud. Overall IRM can provide an additional protection mechanism for enterprise documents that are allowed to be stored in the cloud, yet are not sufficient to control all data propagation.

## 2.3 Information Flow Control

IFC [24, 9] is a mandatory access control model that uses security labels attached to data to control propagation. It ensures that labels propagate with the data during processing, which typically requires modification to operating systems or the end-user applications. *Decentralised* IFC [26] permit the owners of data, i.e. users and applications, to create unique labels and to control data propagation inside a single [13, 26] and across machines [27, 14]. Labels have also been suggested to track the propagation of data within applications to mitigate injection attacks [21, 19] and for monitoring applications for privacy-violating information flows [3].

Despite shown to be effective, using labels to track data flow has often resulted in systems that are not practical. Since labels have to propagate at runtime in a transparent fashion, the resulting overhead makes applications execute several times slower [8]. Some systems achieve better performance but often only target a particular class of applications [13] or require modifications to the execution platform [26, 25].

## 2.4 Data Loss Prevention

Data Loss Prevention (DLP) systems [17, 22] safeguard confidential enterprise data. They typically introduce a proxy that intercepts and analyses all outgoing traffic from the enterprise network. The proxy identifies confidential data that must not leave the enterprise using various methods ranging from pattern matching to file fingerprinting with hashes [11]. While these techniques detect confidential enterprise data, DLP systems are not designed to control data propagation after a file was uploaded to the cloud.

CLOUDFILTER selectively protects specific categories of data (inspired by IRM systems), uses labels based on IFC and checks

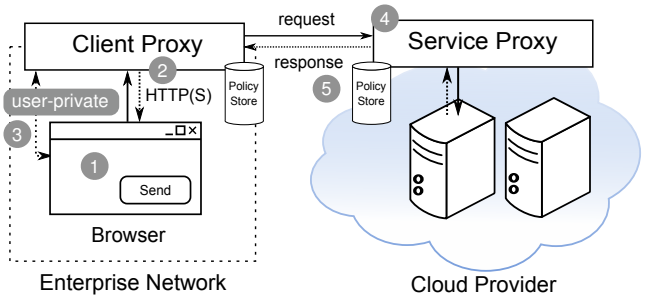


**Figure 1: CLOUDFILTER architecture**

traffic at the boundary of the enterprise network, similar to DLP systems. In contrast to IRM systems, CLOUDFILTER can ensure that users protect the data that they upload. Since labels embedded inside files are left unchanged by typical file operations such as copy or move, CLOUDFILTER avoids the practical issues associated with most IFC systems. With regard to DLP systems, CLOUDFILTER can be seen as an extension, which focuses on monitoring sensitive data after it has left the enterprise network.

# 3. CLOUDFILTER DESIGN

The goal of CLOUDFILTER is to provide an easy-to-use and practical solution for controlling data propagation between an enterprise and cloud services. We focus on cloud storage providers, which represent stored data as files. An important requirement is for the system to be easily applicable across different cloud storage providers with minimal configuration. At the same time, it should be able to adapt to the API of specific cloud storage services. For example, CLOUDFILTER should be able to enforce data propagation policies that prevent users from uploading files to particular folders; such information exists in the HTTP methods invoked.

The need to support existing cloud storage providers imposes a set of requirements on how labels propagate with the data. First, the labels should remain associated with files while stored by the cloud storage provider. Many cloud services allow remote file operations such as copy, move and access to previous versions. Such operations should preserve the labels associated with the files independently of how the operations are implemented. This favours a design, in which labels are *embedded* inside files. Second, labels should remain associated with files if a file is downloaded at an employee's machine. Labels stored locally capture how the file was handled by CLOUDFILTER in the past and such information can be used to reduce the need for user input if the file is uploaded again. In order to maximise the potential for adoption, the mechanism of storing labels should be platform-agnostic. This precludes the use of features specific to file systems (e.g. Extended File Attributes in Linux or Alternate Data Streams in NTFS) that would require translating labels between different representations and may not be supported by cloud storage services. We describe an approach to embed labels in files in §3.2.

CLOUDFILTER consists of the following components (Figure 1):

**Client and Service Proxy.** Two CLOUDFILTER proxies, the client and the service, intercept HTTP traffic between the enterprise and the cloud provider. Each proxy inspects the data being transferred on behalf of the domain that it is part of. In addition, the service proxy may enforce policy specified by the provider's clients. Proxies are responsible for labelling data.

**Policy store.** Each proxy maintains a policy store. The policy store accumulates a set of Event-Condition-Action (ECA) rules. ECA rules specify data propagation policy when a file transfer occurs and therefore control the actions of the proxies. The ECA format is easy to understand and facilitates enforcement when proxies communicate (c.f. §3.1.3).

**Browser extension.** A browser extension collects information used to label uploaded files. It communicates with the client proxy and may explicitly prompt the user for confirmation when the proxy detects a file upload.

Figure 1 illustrates how a file upload occurs between an enterprise and a cloud provider. In step 1, the user submits the file via a web form. The browser plugin attaches to the outgoing HTTP request a set of identification information for the current user along with meta-data about the file, such as the location where the file is stored locally. In step 2, the client proxy intercepts the request, inspects its content and retrieves the user-identifying information. It then matches the request against the ECA rules in its policy store.

When an ECA rule matches a request, the proxy executes an action. In step 3, the action queries the user about the confidentiality of the file being uploaded. The user's reply results in a label that is attached to the file. After the request is logged for future audit, it is forwarded to the cloud service.

In step 4, the service proxy inspects the request along with its labels. The local policy store maintains policy expressed as ECA rules that the cloud provider enforces on client data. For example, a cloud provider may deny hosting sensitive files originating from a particular enterprise to avoid future legal disputes. If the file upload is accepted, the request is forwarded to the storage service.

In step 5, a subsequent HTTP request retrieves the file uploaded in steps 1–4. The service proxy intercepts the response of the storage service and uses the label attached to the data to decide how it should respond to the request. The service proxy may also contact the client proxy to obtain a policy to enforce on its behalf, e.g. to avoid releasing the data to a user who is not located inside the enterprise network.

## 3.1 Specifying data propagation policy

CLOUDFILTER controls data propagation using labels and Event-Condition-Action (ECA) rules. ECA rules encode different policies and the labels attached to files specify which particular policy to enforce when a proxy intercepts a file in transit.

### *3.1.1 Labels*

CLOUDFILTER attaches labels to the files being transferred between domains. Because labels stay associated with files as these propagate between organisations, labels act as a reliable binding of files with the policies that govern their propagation. Labels also store additional meta-data relevant to the enforcement of a particular policy. Since labels encapsulate all the meta-data necessary for policy enforcement, CLOUDFILTER proxies may enforce their designated policies only by inspecting labels, enabling a simple stateless proxy design.

Each label consists of three parts: (1) an identifier, (2) a set of named parameters and (3) the address of the *authoritative* CLOUDFILTER proxy for that label. The identifier is a textual, human-friendly name of a particular data propagation policy. For example, the identifier user-private may be used by an organisation to refer to a policy that restricts the sharing of files in a cloud storage service. The named parameters in a label permit the customisation of the data propagation policy with arbitrary meta-data. For example, the parameter user=[ip108, prp] for the user-private policy further specifies that a file may only be shared between a particular list of users. As will be explained in §3.1.3, an authoritative proxy address ensures the unique binding of the label with a data propagation policy.

| Method | Description |
|---|---|
| `issue` | Issues an HTTP request to its intended destination. Returns the response of the remote service. |
| `return` | Returns an HTTP response to the client the issued the corresponding request. |
| `log` | Logs an HTTP request/response. |
| `getLabels` | Returns the labels of a file stored in an HTTP request, response or HTTP request parameter. |
| `attLabel` | Attaches a label to a file stored in an HTTP request, response or HTTP request parameter. |
| `detLabel` | Detaches a label (similar to `attLabel`). |
| `getContent` | Returns the content of an HTTP message (e.g. method, status code, headers and body). |
| `setContent` | Sets the content of an HTTP message (e.g. method, status code, headers and body). |
| `ask` | Contacts the issuer of an HTTP request to suggest a label for the data it contains (§3.2). |

**Table 1: API available to action scripts**

### 3.1.2 ECA rules

Each ECA rule is triggered when an event occurs. If the condition associated with the event is satisfied, the action is executed.

*Events.*

Since CLOUDFILTER specifically targets cloud storage services where HTTP is the dominant transport protocol, the event that triggers the activation of an ECA rule is the invocation of an HTTP method. A rule may be triggered for incoming HTTP requests that originate from outside a domain or for outgoing HTTP requests invoked from within the domain. An administrator can specify events to be triggered on specific HTTP method invocations or for requests towards particular URIs.

As an example, consider the event:

$e_{uploads}$: {*out*} {*put post*} { `(.*\.)*dropbox.com(/.*)*` }

This event matches all outgoing PUT and POST requests towards the servers of Dropbox. It uses a regular expression to match the HTTP request URI. An enterprise may use such a rule to prevent all file uploads to Dropbox.

*Conditions.*

CLOUDFILTER uses labels as necessary preconditions to trigger actions. Each condition is satisfied by the existence of a labelled file in an HTTP request or response.

Conditions can be *service-agnostic* or *service-specific*. For service-agnostic conditions, the HTTP API of the service is ignored. Such conditions are satisfied by the existence of *any* labelled file in an HTTP request or response. Instead, service-specific conditions require the existence of a label in a particular parameter or part of a request/response. Since HTTP requests and responses may store different files in different parameters and each file may be labelled with a different label, a service-specific condition specifies a set of HTTP parameters and the labels that must be attached to the value of each parameter.

As an example, consider the service-specific condition:

$c_{bank}$: *file* $\Rightarrow$ {*secret*} { `cf\.bank\.com` } .

This condition matches all HTTP requests that contain a parameter called *file* with a value labelled *secret* as defined by the CLOUDFILTER proxy at `cf.bank.com` (again a regular expression). This condition may be used by a service proxy to prevent file uploads that contain secret data.

*Actions.*

An action specifies the data propagation policy of an organisation or cloud service with regard to the class of data that activates the particular rule. Actions are scripts that use the particular CLOUDFILTER API shown in Table 1 and are executed by a proxy.

The three basic operations available to action scripts are `issue`, `return` and `log`. These are used to create, reply to, and store HTTP requests for future audit, respectively. An action script can use these methods to achieve the traditional `allow/deny` semantics of network-level firewalls. The next three methods, `getLabels` `attLabel` and `detLabel` manipulate the labels of the files in transit. Labels are the sole mechanism available to action scripts for storing data across different HTTP requests—scripts themselves are stateless. The decision to attach a label to a file may depend on the actual data being transferred, which are accessed via `getContent`, or on user input using the `ask` method (c.f. §3.2).

### 3.1.3 Distributed data propagation policy

A proxy may obtain and enforce ECA rules on behalf of another proxy. This situation occurs when an enterprise is willing to store data in the cloud, yet it imposes restrictions on how the data may be accessed there. In such scenarios, the client proxy must trust the service proxy and provide it with ECA rules while the service proxy must trust the client proxy as a source of ECA rules.

ECA rule propagation between proxies occurs when a proxy intercepts an HTTP request. Two preconditions must be satisfied: (1) the HTTP request should contain a label that has an authoritative address different from the address of the current proxy and (2) the proxy has no ECA rules with conditions that reference this label. The proxy then directly contacts the authoritative proxy to receive the set of ECA rules that it must enforce. Such *external* ECA rules can only reference in their conditions labels that have the same authoritative address as their origin. This ensures that external ECA rules cannot interfere with the enforcement of other data propagation policies in the same proxy.

In addition, external ECA rules specify a `domain` parameter that restricts the remote proxies eligible to receive them and a `timeout` parameter to ensure freshness. These rules are not used locally—instead they are disclosed to remote proxies upon request. Since ECA rule propagation occurs lazily after an actual file transfer is detected, new policies do not need to be deployed explicitly.

## 3.2 Enforcing data propagation policy

For CLOUDFILTER to be practical, we must minimise the need for user input when attaching labels to files and we should allow for labels to propagate effectively across domains. We have developed an early system prototype, in which the proxy is implemented in Python and the browser extension is a JavaScript add-on for Firefox. We discuss these issues in the context of our prototype.

### 3.2.1 Label Instantiation

Action scripts attach labels to files when a file transfer activates an ECA rule. Four different methods may be used to infer the correct label for the file in transit. First, action scripts can access the file itself. This enables labelling based on the use of specific content in the file to mark security classification, e.g. the use of the word "classified" in documents.

Second, the browser extension attaches contextual information from the user's machine to HTTP requests. Currently this includes the user login, the file location in the file system and its attributes. An action script may use this information, for example, to infer confidentiality for files stored in a particular network location.

Third, files downloaded from a cloud storage service may al-

```
<rdf:Description rdf:about=""
    xmlns:cf0="http://cloudfilter.doc.ic.ac.uk/0">
   <cf0:domain>cf.doc.ic.ac.uk</cf0:domain>
   <cf0:id>user-private</cf0:id>
   <cf0:parameters>
      <rdf:Seq>
         <rdf:li>user</rdf:li>
      </rdf:Seq>
   </cf0:parameters>
   <cf0:user>ip108, prp</cf0:user>
</rdf:Description>
```

**Figure 2: An CLOUDFILTER label encoded using XMP**

ready contain labels from previous interactions with CLOUDFILTER. Such labels are useful if a file is then re-uploaded: they can be used to avoid querying the user. To ensure that no decision is taken based upon labels that are no longer valid, we can store file hashes inside labels and detect file modifications.

Finally, the client proxy can ask the user for input during upload (i.e. using the `ask` method in Table 1). The browser extension receives an HTML form generated dynamically by the action script, displays it in an overlay to the user and sends the response back to the proxy. The form content depends on the policy being enforced.

### *3.2.2 Label Propagation*

In order to embed CLOUDFILTER labels in files, we can leverage the concept of meta-data that many file formats support. In our prototype, we use Adobe's Extensible Meta-data Platform (XMP) [1]. XMP is a specification for representing arbitrary meta-data in RDF/XML and storing it inside various file formats. There exists an SDK for programmatically embedding XMP meta-data in multiple popular file formats (e.g. PDF, EPS and JPEG). Figure 2 shows how the label user-private from §3.1.1 can be represented in XMP.

We also use XMP to store labels inside Microsoft Office documents. Office file formats after version 2007 conform to the Open Packaging Conventions [7]. Each package (e.g. a Word document file) is an Zip archive of both text (e.g. XML) and binary (e.g. images) files that are known as *parts*. Application-specific properties, such the total character count in a Word document or XMP-encoded CLOUDFILTER labels, can be stored in additional parts inside the package. Since Office applications ignore parts of unknown document types, XMP labels do not incur compatibility issues.

## 4. APPLICATION USE CASES

For CLOUDFILTER to be an effective mechanism for controlling data propagation, it must be applicable to representative cloud storage service and be capable of enforcing useful data propagation policies. This section presents representative policies for two such systems, Dropbox and GSS, and we discuss their limitations.

### 4.1 Dropbox

Dropbox [4] is a cloud storage service that has popularised the concept of online storage. Typically, it uses a native client to achieve transparent synchronisation. The service supports an HTTP API and exposes a web interface with most of the native client's functionality. Since the native Dropbox client does not rely on the HTTP API [15], the use of Dropbox is supported by CLOUDFILTER only via the web interface.

The simplest policy that can be enforced by CLOUDFILTER is:

**Policy 1:** Prevent the upload of any enterprise files to Dropbox.

This policy, albeit restrictive, is nevertheless an improvement over a network-level blocking of Dropbox. Employees maintain access to personal files, which may be useful while at work, and they are able to receive files from external collaborators. The ECA rule to capture this policy is:

| Event | Condition | Action |
|---|---|---|
| $e_{uploads}$ (§3.1.2) | – | `return("403")` |

A second, more permissive policy is:

**Policy 2:** Only allow uploading public documents to Dropbox.

To enforce this policy, a more elaborate action script is required to infer when a document is considered public. A simple example that repeatedly queries the user is:

| Event | Condition | Action |
|---|---|---|
| $e_{uploads}$ | – | `form=createHTMLForm()`<br>`resp=ask(form)`<br>`if resp=="public":`<br>`  log()`<br>`  return(issue())`<br>`else:`<br>`  return("403")` |

Policies 1 and 2 can be expressed as ECA rules without considering the Dropbox HTTP API. Their enforcement depends on the proxy's ability to detect all file transfers over HTTP. Using HTTP files may be transferred (1) in the body of an PUT request, (2) in the body of a response to a GET request or (3) via POST as a result to an HTML form submission. Since binary file upload via POST is standardised [10], the proxy can reliably intercept files in transit. Therefore the same ECA rules that implement Policy 1 and 2 can be used for a different provider without modifications.

### 4.2 GSS

GSS [6] is an open-source cloud file storage system used in Greece to offer public university staff and students 50 GiB of free online storage space. The system supports many advanced features, such as versioning, file sharing and full-text search. It can be accessed via a web client over HTTP and using WebDAV, i.e. an extension to HTTP for file storage in the web [5].

A policy that a participating university $U$ may declare is:

**Policy 3:** Ensure that all documents uploaded from university staff are only shared between staff of the same university.

This policy can reduce the scope for disclosing sensitive information to student users of the service. To enforce it, a proxy in $U$ attaches to files uploaded by members of staff a label with the identifier $U$-confidential. It also stores an external ECA rule. Assume that such a file is shared with a student user of GSS. If the student user tries to download the file, a CLOUDFILTER proxy deployed in the network where GSS is installed observes the label and fetches the external ECA rule from $U$'s proxy. The action script identifies that the request came from a student user and aborts it.

Key to enforcing Policy 3 is collecting contextual information about the documents being uploaded. CLOUDFILTER's ability to adapt to the API of a particular service is beneficial because the GSS API uses usernames and university domains as part of file URIs. ECA rules can thus reliably obtain the required information by accessing the HTTP requests directly without input from the browser plugin. For GSS, this means that Policy 3 can be enforced even for non-browser, WebDAV-based access.

### 4.3 Limitations

A limitation when enforcing Policy 2 is the need for user input when deciding on labels. A strict policy that repeatedly prompts the user before every file upload will gradually become less effective because users will ignore the prompts. If a provider does not have a

CLOUDFILTER proxy installed, policies that restrict downloading (e.g. Policy 3) cannot be enforced.

All the above policies can only be enforced for a fixed set of file formats. For CLOUDFILTER to be effective, it must be able to embed labels for all file types used in a particular environment. Ideally, this can be achieved by using native meta-data support of many file formats or, in the worst case, by encoding labels as data.

CLOUDFILTER should also be able to intercept and decrypt network traffic. To achieve this, all HTTP traffic between enterprise clients and cloud storage services should be routed via the CLOUDFILTER proxies. Proxies can handle HTTPS traffic by acting as a man-in-the-middle using—in the client case—certificates added to the browser's trust set. The company firewall should deny direct client communication with the cloud storage services to prevent unmonitored file transfers via unsupported application protocols.

In practice, users may connect directly to the service provider bypassing the CLOUDFILTER client proxy if they are able to connect to arbitrary networks or administer their own machines. Malicious users and providers may also circumvent proxies: they can thus abuse HTTP for file transfers. However, we believe that (1) it is realistic to assume control of client equipment in scenarios when data propagation is important and (2) cloud storage providers have an incentive to support the operation of CLOUDFILTER.

Overall CLOUDFILTER is a mechanism to control data propagation to the cloud but does not make cloud storage services more secure. An attacker who compromises a cloud service may still transfer arbitrary data, for example, by removing labels from files.

# 5. CONCLUSIONS

In this paper, we proposed a practical method to control the propagation of data between enterprises and cloud storage services. We showed that the use of HTTP exposes an interception point at which traffic can be inspected reliably. We also suggested a method to specify data propagation policy and shown that it can capture relevant policies for enterprises that consider migrating to the cloud. We believe that our approach can promote the use of cloud storage in enterprise environments.

We plan to build upon this research and explore different types of cloud services apart from storage. Open challenges include how to correctly propagate labels from input to output if a service does not use a file abstraction. We want to explore whether service providers themselves can provide us with such information.

Regarding our prototype implementation, we plan to explore different isolation strategies (e.g. PyPy's sand-box [20]) to ensure that external action scripts do not interfere with the correct operation of proxies. We will also investigate embedding XMP labels to more popular document formats. Another worthwhile direction is to integrate our work with provenance-aware file systems [16] in order to improve CLOUDFILTER's ability to infer labels.

Finally, we want to explore the performance impact of our system. To improve performance, we plan to integrate a CLOUDFILTER proxy with an open-source storage system such as GSS. With access to the storage service we can identify any labels embedded in each file once and then supply these labels to the proxy during download. The proxy can then use such pre-identified labels to avoid the cost of checking every file in transit for labels.

**Acknowledgements.** This work was supported by grant EP/J020370 ("CloudFilter: Practical Confinement of Sensitive Data Across Clouds") from the UK Engineering and Physical Sciences Research Council (EPSRC).

# 6. REFERENCES

[1] Adobe. XMP Specification Parts 1-3, 2012.

[2] S. Akhshabi and C. Dovrolis. The evolution of layered protocol stacks leads to an hourglass-shaped architecture. In *SIGCOMM*, Toronto, Canada, 2011. ACM.

[3] J. Dongseok, R. Jhala, et al. An empirical study of privacy-violating information flows in JavaScript web applications. In *CCS*, Chicago, IL, 2010. ACM.

[4] Dropbox Website. `http://www.dropbox.com`.

[5] E. Dusseault. HTTP Extensions for Web Distributed Authoring and Versioning (RFC 4918). *IETF*, 2007.

[6] GSS Project. `http://code.google.com/p/gss/`.

[7] ISO/IEC 29500-2:2011. Office Open XML File Formats Part 2: Open Packaging Conventions, 2011.

[8] V. Kemerlis et al. libdft: Practical dynamic data flow tracking for commodity systems. In *VEE*, London, UK, 2012. ACM.

[9] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the Linux operating system. In *ATC*, Boston, MA, 2001. USENIX.

[10] L. Masinter. Returning Values from Forms: multipart/form-data (RFC 2388). *IETF*, 1998.

[11] McAfee. Demystifying Data Loss Prevention, 2010.

[12] Microsoft. Information Rights Management in SharePoint Foundation, 2010.

[13] M. Migliavacca, I. Papagiannis, D. Eyers, B. Shand, J. Bacon, and P. Pietzuch. DEFCon: High-performance event processing with information security. In *ATC*, Boston, MA, 2010. USENIX.

[14] M. Migliavacca, I. Papagiannis, D. M. Eyers, B. Shand, J. Bacon, and P. Pietzuch. Distributed middleware enforcement of event flow security policy. In *Middleware*, Bangalore, India, 2010. ACM/IFIP/USENIX.

[15] M. Mulazzani et al. Dark clouds on the horizon: using cloud storage as attack vector and online slack space. In *Security Symposium*, San Francisco, CA, 2011. USENIX.

[16] K. Muniswamy, D. Holland, et al. Provenance-Aware Storage Systems. In *ATC*, Boston, MA, 2006. USENIX.

[17] MyDLP. `http://www.mydlp.com/`.

[18] Oracle. Information Rights Management Data Sheet, 2010.

[19] I. Papagiannis, M. Migliavacca, and P. Pietzuch. PHP Aspis: using partial taint tracking to protect against injection attacks. In *WebApps*, Portland, OR, 2011. USENIX.

[20] PyPy's sandboxing features. `http://doc.pypy.org/en/latest/sandbox.html`.

[21] E. Schwartz, T. Avgerinos, and D. Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Security & Privacy*, Berkeley, CA, 2010. IEEE.

[22] Symantec Data Loss Prevention. `http://www.symantec.com/data-loss-prevention/`.

[23] Symantec. State of cloud survey: Global findings, 2011.

[24] US Department of Defense. Trusted Computer System Evaluation Criteria (Orange Book), 1983.

[25] A. Yip, X. Wang, et al. Improving application security with data flow assertions. In *SOSP*, Big Sky, MT, 2009. ACM.

[26] N. Zeldovich, S. Boyd, et al. Making information flow explicit in HiStar. In *OSDI*, Seattle, WA, 2006. USENIX.

[27] N. Zeldovich, S. Boyd, and D. Mazières. Securing distributed systems with information flow control. In *NSDI*, San Francisco, CA, 2008. USENIX.