Proceedings of the

Second EECE 512 Mini-Conference on Computer Security

Vancouver, BC, Canada
April 10 - 12, 2007

Sponsored by

Laboratory for Education and Research in Secure Systems Engineering

Technical report LERSSE-TR-2007-03
http://lersse.ece.ubc.ca

Table of Contents

Conference Organization

General Chair: Konstantin Beznosov

Program Chair: Jeremy Hilliker

Session Chairs: Maryam Najafian Razavi
Brett Cannon

Proceedings Chair: Rodrigo Werlinger

Controlling Access to Resources Within The Python Interpreter

Brett Cannon
University of British Columbia
brett@python.org

Abstract

Version 2.5 of the Python programming language contains no mechanism for restricting access to resources by Python code. This is a slight hindrance to the language as it is used in many situations, such as a domain-specific language in other applications, where some mechanism to control what resources Python code can access would be helpful.

Python did once have a security mechanism for restricting resource access, but it was disabled in version 2.3. The disabling of the security mechanism was driven by a lack of security expertise on the part of the Python development team. This means that any introduced security mechanism should, if possible, not require language support so as to prevent the need to turn off any new security mechanism in the future.

This paper presents a security mechanism whose impact upon the Python language is minimal. By removing four function or methods from Python's built-in namespace and utilizing Python's modularity in terms of its connection with its underlying interpreter, the proposed security mechanism has minimal impact upon the language. The mechanism allows for controlling access to resources within a single Python interpreter. This allows Python to have some form of a security mechanism between Python code and the system it is running on.

1. Introduction

The Python programming language [32] has a wide range of uses. It is used for teaching computers science [27], as a domain-specific language (DSL) [31], and for developing large applications [6]. Part of this popularity stems from the fact that Python is an interpreted language with its own virtual machine.

Something that all users of Python have in common is trust that the code they are executing in the Python interpreter is not malicious. The current version of Python (version 2.5) lacks any mechanism to enforce a security policy related to controlling access to resources (as defined by [11]). While some informal methods exist, Python lacks any mechanism built into the language or its implementation to prevent code from accessing resources such as files or sockets.

The lack of a way to restrict resource access is unfortunate because having a way to enforce a security policy for the Python interpreter would be useful in several situations. When Python is embedded in an application as a domain-specific language (DSL), it would be helpful to make sure that users of the application can feel confident that some source code they download from the Internet for use will not delete files from their file system that it should not have access to. In the educational realm making sure students do not accidentally harm their own computers would help alleviate worries of a programming mistake causing serious damage.

For these reasons and various other ones, this paper addresses work done to introduce the needed mechanisms to control access to resources from within the Python interpreter. The contribution of this paper is to present an approach to introducing a mechanism into Python that allows for controlling access to resources that does not require changes to the Python language itself and only minor changes to some select object types.

2. The Python Programming Language

The Python programming language is an object-oriented, interpreted language that has uses in a wide range of domains such as scientific applications, web sites, and desktop applications [5]. Systems programming is essentially the only are where Python is not currently in use.

Python supports both the procedural and object-oriented programming paradigms along with some functional support. Python's embrace of object-oriented programming is deep. Within Python everything is an object; numbers, strings, functions, etc. There is complete support for operator overloading by objects. There is also deep support for introspection capabilities on objects. For instance, every object stores a reference to the class or type that the object is an instance of (stored in the `__class__` attribute).

One of the basic tenants of Python is that "Explicit Is Better Than Implicit" [26]. This has led to Python being designed so that most actions taken within the language have an explicit representation through some particular object. The semantics are also explicit. A good example of this explicitness is the promotion within Python of modularity amongst source files.

In Python a file that contains source code is called a module. It is a common practice to reasonably separate source code into individual modules, where semantics makes such a separation reasonable. In order to bring a module into another module's namespace an explicit statement must be made using the `import` keyword. Specifying a module with the `import` keyword "imports" the module such that the code in that module is then available to the code that executed the import (e.g., "`import foo`" brings the `foo` module into the current namespace and makes the objects defined within the module accessible under the `foo` name).

But what makes the `import` keyword a wonderful example of Python's explicit modularity is how the statement works. The mechanism that implements the `import` keyword for interpreting a file and creating an object representation is handled by the `__import__` function. Defined in the built-in namespace that is accessible by all executing code, the function is passed the name of

the module being requested for importation and returns the object representation of that module: "`__import__('foo')`".

The `import`/`__import__` dichotomy also serves as a good illustration of how Python's interpreted nature ties into the language. Python itself exploits the fact that it is an interpreted language by allowing certain language semantics to be configurable based on the interpreter as it is running. The interpreter is not a blackbox that one cannot change but a part of the larger system that can be modified in certain ways to play a role in how Python code is executed.

By providing a rich C API to support tying into the Python interpreter, Python has garnered great success as a DSL in various programs implemented in C. The C API also allows for modules to not only be implemented in Python source code but also in C.

3. History of Security in Python

Currently Python lacks a mechanism for enforcing a security policy; this was not always the case. Added to Python in 1995, the `rexec` module [7] (along with the related `Bastion` module introduced in 1996) provided the ability to run source code in a sandbox. Following the design of Safe-Tcl [23], `rexec` used Python's `exec` statement to run code with a restricted built-in and global namespace. Support was also included in the implementation of Python's interpreter in the form of flagging execution frames that had a differing built-in namespace from that of the interpreter as being in a "restricted" mode. This restricted mode turned off certain introspection capabilities deemed dangerous. This mechanism was also available at the C level through Python's C API.

The `rexec` module provided the means of enforcing a security policy when executing Python code for many years. But starting in version 2.3 of Python, both the `rexec` and its companion `Bastion` module were disabled. This stemmed from the key fact that there were (and still are) not any security experts among the various developers of Python.

Because Python is an open source project the only people who help develop Python are volunteers. An issue that stems from Python's open source development is there is no way to guarantee that any member of the development team is an expert in security as someone would need to volunteer to fulfill that role. Having no security expert on the development team can cause issues as someone might make a change to the language that breaks security but is not caught by anyone on the development team. The realization that the security mechanism in place was being unintentionally broken happened during the development of Python 2.3 and led to the disabling of `rexec` and `Bastion` until a developer joined the development team who had the proper expertise on security [33].

4. Related Work

The work presented by this paper is obviously not the first attempt to introduce a mechanism to enforce a security policy within a programming language. As stated in Section 3, Safe-Tcl [23] was the inspiration for Python's first security mechanism. The approach taken in Safe-Tcl is to allow multiple interpreters to execute within the same process. Each interpreter has a security policy enforced by controlling what commands are available in the global namespace. By creating the proper delegates an interpreter can be provided commands that have restricted abilities as the programmer deems fit.

Perl provides two types of security. One is a taint mode on all string inputs [25]. Tainting provides for information-flow security by flagging all incoming data as "tainted" [36]. This allows Perl's interpreter to disallow tainted data to be used in any place that is defined unsafe (e.g., writing to a file or socket, command to the operating system, etc.).

The other type of security in Perl is the `Safe` module [10]. The module restricts the namespace that code will be run in. The code being restricted is then compiled to only execute within the namespace specified to restrict the abilities of the code in question.

CaPerl is a modified version of the Perl language that implements a capability system within the language [18]. By introducing new keywords into the language that specify what is trusted and what is not one is able to restrict what abilities are exposed to various pieces of code. It is based on the ideas of capabilities which are discussed later in Section 5.3.

Ruby's security is much like Perl's taint system [30]. The significant difference between Ruby and Perl, though, is that Ruby implements levels of security for its taint model. The taint levels allow certain operations to allow tainted data while others don't based on how restrictive the security level is meant to be.

JavaScript provides several means to securely run code when used in a web browser [3][9]. One is using access control to enforce the security policy. By using access control lists (ACLs) a script is restricted to accessing web pages that are listed in that code's ACL. JavaScript as run in a web browser can also have signed code. This provides guarantees that if you trust the person who signed some code that it was unmodified and thus as trustworthy as the person who signed it.

Java has had its security implementation evolve over the life of the language. Originally, Java followed a sandbox model where a class loader enforced a local namespace so that untrusted code could not influence other code [14]. The sandbox model did not allow for fine-grained control and thus was changed [17]. With Java 2, a security policy was possible to be set as represented by the `java.security.Policy` object. Security domains were also introduced with each class being a member of a single domain. This allowed each object to be connected to the security policy that was associated with the security domain. To enforce this, the execution stack is traced to make sure that the call will not violate the security policy [8]. Java code can also be signed like JavaScript for use in a web browser so as to allow execution of code from trusted sources without worrying about malicious attacks [13].

The .NET framework builds off of Java's general design and attempts to fix issues that have crept up over the years with Java's security system [24]. Like Java, .NET verifies that the file format of the code to be run is safe. While both use policies defined externally to the the respective runtimes, .NET uses the intersection of groups of policies specified. Both use an analysis of the execution stack to make sure that a call is safe. .NET can do some static security policy enforcement, unlike Java. There is also the ability to sign code to verify it comes from a trusted source.

Programming languages are not the only way to provide security. Operating systems must also specify security policies on multi-user systems. Probably one of best known instance of an operating system's security mechanism is FreeBSD's "jail" facility [16]. By altering the view of the system a user has, the jail facility helps to prevent a user from accessing certain resources. This alteration of view even affects the root account for the jail [19]. Further enforcement is through proper user checks in system calls that can perform possibly malicious actions on behalf of the user.

5. Design

5.1 Criteria

From the outset, the design of the security mechanism to be implemented for Python has two design constraints. The first one is that the language could not substantially change in terms of feel. Within the Python community, the term "Pythonic" embodies this idea. Anything deemed Pythonic is considered to follow the general philosophies of the Python programming language. While an

extremely nebulous definition, the closest to an "official" definition of the term can be found in [26]. Adding static type checking, for instance, would not be Pythonic as it goes against Python's promotion of interface typing.

The restriction of not introducing any features to the language that could be deemed not Pythonic has meant that any change visible at the language must first be deemed Pythonic. This is critical in order for current Python programmers to have psychological acceptability for the security mechanism and are comfortable with it [28][35]. If the security mechanism deviates too much from common Python practices it would be rejected by current users, severely hindering its acceptance.

The second restriction placed upon the design is that it cannot hinder Python's development team. This stems from lessons learned in Python's history with security as discussed in Section 3. The developers of Python should not have to shoulder a great burden in order to continue to support any security mechanism that is designed and implemented. By making it a requirement of the design that the Python developers do not have to worry about security in every decision they make, there are two benefits. One is that the developers are allowed more freedom in changing the language; minimizing any inconvenience or hindrance lets Python grow and evolve more easily. Second, by not forcing the developers to constantly make sure that their work won't break any security mechanism, the possibility of a vulnerability being introduced becomes minimized. This helps prevent a reoccurrence of security holes like what happened with the `rexec` module.

An unfortunate side-effect of these design criteria is that deeply integrated security solutions are not acceptable. This makes the solutions taken by Java and .NET not directly applicable in this situation as both languages were designed with security in mind and thus started with their security mechanism integrated into the implementation. Both Java and .NET take an approach where a class loader is directly integrated into their interpreter and that is not acceptable in this instance as that would burden the Python development team, something that this design is trying to avoid.

5.2 Threat Model

The purpose of the work under discussion is to handle a threat model where arbitrary Python code is run in a single Python interpreter without fear that it is able to gain access or control of resources it is not explicitly given (Python supports running multiple interpreters within a single process, but this work only addresses the case of a single interpreter running). In terms of what kind of adversary the security mechanism is meant to guard against, there are two. One is where an adversary misrepresents what a piece of Python code does in terms of accessing system resources such as files or sockets. A trusting user then runs the code from the adversary which then accesses resources that the code was not expected to access. The access can include reading, writing, or destroying resources. The accessing could be purely to destroy other resources or could be to steal sensitive information.

The other kind of adversary is one where a programmer happens to accidentally destroy a resource. This might come in the form of a person new to programming accidentally overwriting a file when they meant to open for reading or to write to a new file path instead of an existing one.

With those types of adversaries being considered, the Python interpreter cannot gain access to resources that the process the interpreter is running in possesses without explicitly being given access to those resources. The operating system bestows and restricts the resources a program can access upon individual processes. This makes the process, from the program's point of view, the powerbox; the object that has the greatest amount of power and that retains the power to give out abilities and resources [29]. If the powerbox did not regulate what resources the interpreter had at its disposal then it could do anything the Python process could do.

Related to the process acting as the powerbox, the interpreter cannot gain access to the resources of the operating system without explicitly being given them. This issue is mediated by having to go through the powerbox.

In the end, this work is trying to regulate the authority the Python interpreter has upon various resources that can possibly be granted to it by the operating system. By "authority" we mean the "effects a program may cause on objects it can access, either directly by permission, or indirectly by permitted interactions with other programs" [20]. This means we are trying to mitigate the threat posed by code run in a Python interpreter by controlling what resources are exposed to and by the Python interpreter.

5.3 Object-Capabilities

In general there are two major types of security for controlling access to resources, "list-oriented" and "ticket-oriented" approaches to security, which are more commonly known as ACLs and capability systems, respectively [28]. The former uses a list of authorized users to control access to resources. Code that is running would be represented as a user, and if the represented user is not listed as authorized to use a resource it would be denied access. A capability system uses tokens or tickets to represent the right to access a resource. When a ticket is presented to gain access to a resource then the code is granted or denied access based on whether that ticket was valid for approving the right to use a resource.

An analogy one might use is trying to gain access to a night club where a special event is occurring. With ACLs, a bouncer at the door would have a list of people authorized to attend the event. By presenting identification establishing you are who you claim, the bouncer can check your name against the list and either permit or block your entrance. A capability system would be based on invitations that every attendee was sent. If someone came to the door and presented a legitimate invitation they would be granted access.

A type of capability system that exists is object-capabilties [22]. This type of security model as implemented in an object-oriented programming language "uses the reference graph as the access graph" [20]. Authority to use a resource is granted based on whether a reference to an object that possesses the authority to access the resource is available. This makes object-capabilities' security based on the controlling of references to objects.

In order for object-capabilities to work, three things must be in place in the language: references cannot be forged, immutable shared state, and private namespaces [22]. Unfortunately Python only has the first of the three abilities by default within Python itself. The latter two have solutions and are addressed in Section 5.4.

With the basic conditions met for using object-capabilities, the question becomes why object-capabilities over another approach? The answer comes from Python's dynamic, modular design along with not wanting to modify the Python language if it can be avoided. Object-capabilities can be implemented on a per-object instance. A single function can implement object-capabilities if so desired without requiring every function to use the system as long as the three required features of object-capabilities is met. This case-by-case implementation approach allows for only parts of a system to bother with security. This works well in the situation under discussion as it minimizes what needs to be changed within Python. No huge re-architecture or addition of security needs to be added to Python but only to individual objects where security matters.

And as object-capabilities allows for a per-object implementation it also alleviates tying into the language directly if it is not desired. In the case of ACLs, the language itself needs to understand

ACLs in terms of identities and the proper propagation of identities. In both Java and .NET, for instance, the security mechanism is directly integrated into the interpreters of the two systems. However object-capabilities requires no such thing if the three implementation requirements are met. A function can implement its own token system and decide itself how it will handle those tokens. It is optional for a language to participate in the security mechanism with object-capabilities.

In general there are two problems commonly believed to exist with any capability system along with one weakness [11]. One perceived problem is that of confining capabilities so that they do not improperly propagate (e.g., authority is given to an object that then hands to another object where it is not desired that the second object gain the authority being passed around). In general there is a known solution [21], but in the case of the threat model being considered in this paper it is not an issue. As there is only a single interpreter there is no one to improperly propagate authority to. The powerbox which gave the interpreter the authority in the first place is the only other party even under discussion and the powerbox already has all authority that the interpreter has and then some. There is no other interpreter that could be improperly given authority for something by some other interpreter.

Another perceived problem of a capability system is that of the revocation of rights. This is a consideration for our threat model. If the powerbox decides to revoke the authority to access a resource after it has been granted (e.g., a one-time access to a resource) there needs to be a way to revoke that right. A solution exists through delegating authority through an intermediary [21]. As will be discussed in Section 5.4, there is a way to implement a delegate in Python, albeit not in Python source code but in C code.

Finally, the weakness that capabilities have is in answering the question, "given an object, what subjects can access it" [11]? But in the case of our threat model, this is once again not an issue. As we have only a single interpreter to consider within the powerbox there can only ever be a single subject whose authority is being restricted. This singularity of only having a single interpreter running makes the plurality of "subjects" meaningless in regards to the question being posted and thus a non-issue.

5.4 Securing a "Bare" Interpreter

As a starting point for discussing what must be done to secure Python, consider what could be deemed a "bare" interpreter. When the interpreter has been initialized and no code has been executed within the interpreter it can be called "bare". If the interpreter cannot control what resources are exposed within the interpreter in this bare state then there is no chance of regulating access to resources as a bare interpreter is essentially the initial state the interpreter is in when it begins executing code.

With no code imported, an interpreter starts off with the built-in namespace created. The namespace contains various things considered essential or highly useful for almost all Python programs. Exceptions, the various built-in data types, and useful functions are contained within the built-in namespace. Controlling this namespace, along with the global namespace of running code, is important as capability systems in general rely on controlling namespaces [15].

Because the built-in namespace is exposed in all Python code, it must be made safe such that objects in the namespace that indiscriminately give access to resources are removed or modified. Types must either be changed so that one cannot instantiate new instances indiscriminately from them or be removed completely. Methods on the various types in the built-in namespace might need to be removed or modified if they have the authority to access resources that should be protected. The same possibility or modification or removal applies to functions. Table 1 lists the various methods and functions that need to be removed from the built-in namespace.

In terms of functions, the `execfile` and `open` functions need to be removed. The former takes a file path to a Python source file and executes it in the interpreter, returning an object representing the code that was executed. This is dangerous as it allows access to the file system indiscriminately. The `open` function has the authority to open files for reading or writing. For the same reason as `execfile` needs to be removed, the `open` function needs to be taken out of the built-in namespace; it has authority to access the file system indiscriminately. It is possible to provide a delegate to allow for restricted use of these functions; such a delegate is discussed in Section 6.3.

Two data types in the built-in namespace that need to be changed in order to prevent indiscriminate instantiating of them are the `code` and `file` types. For both types the way to prevent instantiation is to remove their `__init__` methods as this is the method Python calls on a class or type when it creates a new object instance.

The `code` type allows one to create code objects, which represent executable Python code. The type needs to restrict indiscriminate instantiation as it is possible to create `code` objects with Python's bytecode without going through Python's internal compiler. Since Python lacks any way to verify the soundness of bytecode this becomes a possible denial-of-service (DoS) attack by crashing the interpreter. An escalation of abilities might be possible through some very specifically crafted bytecode, although this is pure speculation.

The other type that must be prevented from being directly instantiated is the `file` type. As the object representation of files on the file system, unchecked instantiating of the type would lead to uncontrolled access of the file system. By removing the ability to instantiate the `file` type directly all file access can be redirected through a delegate (such as a delegate for the `open` function). This provides a single access point for the file system within the built-in namespace which makes for simpler control of file system access.

In a bare interpreter, there is a single method that causes Python to not meet the immutable shared state requirement of object-capabilities (as mentioned in Section 5.3). The `__subclasses__` method on the `object` type returns a list of all subclasses. By virtue of the `object` type being the root type of all objects in Python the `__subclasses__` method returns *all* classes defined by any Python code within the Python process, not just within the interpreter as some objects are created to allow the interpreter to work. Without the removal of this method it would not be possible to prevent access to a class or type.

5.5 Securely Allowing Imports

The usefulness of code that runs in a bare interpreter is fairly limited. Without the ability to import external code almost all useful functionality would need to be re-implemented by the code running in the interpreter. Obviously not being to import code would be a great hindrance on programs and cripple the usefulness of Python.

A basic overview of importing modules is covered in Section 2, but a more thorough discussion is called for to understand how imports need to be changed. There are five different kinds of modules that vary based on how the code is implemented and stored. One is built-in modules which are implemented in C and compiled into the Python executable. "Frozen" modules are Python bytecode stored in C code that is then compiled into the executable. Extension modules are written in C and compiled into external object files. Python bytecode (which typically has a `.pyc` or `.pyo` file extension) are byte-compiled versions of Python source code. Python source modules are written entirely in Python source code (and typically has a `.py` file extension).

To remove	Type of object	Purpose of removal
object.__subclasses__	method	Exposes all defined classes. Removal creates immutable shared state.
open file.__init__	function method	Allow unmitigated read and write access to files. Also exposes information about existence of files.
execfile	function	Allows access to any Python source file. Can also be used to query about existence of files.
code.__init__	method	Can create malicious bytecode which is never verified to be well-formed.

Table 1. Built-in methods and functions to remove from Python's built-in namespace.

Regulating what modules can be imported is critical to controlling what resources Python code has access to. With a bare interpreter a safe environment is provided. By controlling imports the environment can stay safe for use as imports are the only way to introduce code that has authority to perform actions that are not available within a bare interpreter.

Regulation of what modules may be imported is done based on the kind of module. Python source code and bytecode modules, in isolation, have only the abilities provided by a bare interpreter. Modules implemented in C, though, can have the same abilities as C code, which is essentially the ability to do anything. By blocking or carefully controlling modules that are implemented in C code then only those modules that can be trusted can be allowed to be imported and thus regulate what abilities the interpreter has access to.

Controlling built-in, frozen, and extension modules is critical to prevent unmitigated access to executing C code. A whitelist should be implemented for each type of module so that only those modules deemed safe through a security audit may be imported. By making the whitelist on a per-type basis instead of based on name alone allows alternative implementations of a module's interface to be implemented by another kind of module (e.g., implementing an extension module's interface in pure Python source code).

Python bytecode files should not be allowed for any reason. They are considered an optimization and not a requirement for the Python interpreter to be able to import Python source modules. They pose a possible risk through ill-formed bytecode by either crashing the interpreter or gaining escalated rights. Theoretically trusted bytecode should only be written to the file system by the Python interpreter, but other programs could be running on the same system that attempt to create ill-formed bytecode and trick the interpreter into trusting the bytecode.

With imports properly controlled so that potentially dangerous modules cannot be accessed, and the bare interpreter considered safe, any Python source code should be trusted to run safely. The bare interpreter provides a secured base for any pure Python source code. By restricting imports the protected base can be safely extended through modules without losing its security.

5.6 Discussion

In what has become considered a seminal paper on computer security [28], eight design principles are laid out that any security mechanism should strive to follow.

First, one should drive to keep a design simple: "economy of mechanism". Overall the design presented in this paper is straightforward. Making the bare interpreter safe consists removing some functions or methods. The greatest complexity is adding support for regulating import based on the kind of module being requested. But since the import machinery is an integral part of the Python language it is simple to test any modifications by running all of Python's regression test suite with any changes implemented. Compared to `rexec`, the new design is simpler since no direct support is needed within the interpreter.

"Fail-safe defaults" dictates that what is to be allowed by a security policy should be explicit, not implicit based on what is **not** allowed. By excluding potentially dangerous types and functions from the built-in namespace, the bare interpreter by default is protected. And by using whitelisting instead of blacklisting for imports the default security is to deny instead of allow importing possibly dangerous modules.

By having "complete mediation" all access to objects need to be checked for authority. Object-capabilities mediate this to reference access. Because there is no explicit check of authority, this principle does not directly apply. Indirectly, object-capabilities performs a "check" every time a reference is used since references cannot be forged in Python and thus any use of a reference is an implicit clearance to use it.

Because Python is open source software, the language itself has an "open design". The security mechanism proposed in this paper does not rely on any hidden value or implementation detail. Auditing of the security mechanism is possible for anyone and is encouraged.

There is no need for a "separation of privilege" with this security mechanism. There is only a single interpreter in a single process being considered. The concept of multiple owners of a Python process does not exist. This also applies to the principle of "least common mechanism".

The concept of "least privilege" is left up to the user through the whitelisting of modules. But by using whitelisting over blacklisting users are not given overly broad privileges by accident.

"Psychological acceptability" should be very high for this design. No new syntax or concept has been introduced into Python. The greatest change is that of whitelisting of certain type of modules. While this is a change to how Python normally operates, it will manifest itself in the normal error of an import error. Whitelisting could be viewed as specifying what is installed in the interpreter and verifying a program can run is like checking installation requirements; nothing new to programmers.

6. Implementation

Since the security work occurs outside of the interpreter, most of the implementation details are in C code. As such, unless otherwise noted, any code mentioned relating to implementation details should be considered in C code.

6.1 Built-In Functions

Removing the built-in functions `execfile` and `open` is straightforward. Python's built-in namespace is a dictionary stored on the C struct that represents the interpreter (i.e., the `builtin` field of the `PyInterpreterState` struct). Removing the functions from the dictionary is a simple call to the `PyDict_DelItem()` function to delete the items from the dictionary.

6.2 Built-In Types

To understand how the `file` and `code` types have been altered one needs to understand how Python implements types at the C level. In C code the `PyTypeObject` struct represents a Python type. Its

various fields represent internal data along with functions that are to be exposed in Python as methods. Two such functions are those that are stored in the `tp_new` and `tp_init` fields and are exposed in Python as `__new__` and `__init__` methods. The `tp_new` function is used to allocate the memory for any new instance of the type along with putting the instance in a valid state. The `tp_init` function is what takes the arguments passed in during the instantiating call to the type and creates the proper state within the instance based on those arguments.

For both the `file` and `code` types the `tp_init` fields in `PyTypeObject` have been set to `NULL`. This is the equivalent of removing the `__init__` methods from the types. Leaving the `tp_new` fields alone for both types allows for either type to be instantiated but not be initialized in any specific way that exposes the authority to access resources.

Both types need a way to initialize new instances within C code, though. For `code` objects the capiPyCode_New() function already exists for this purpose. For the `file` type, the `PyFile_Init()` function has been introduced. This allows the `open` built-in to Python, at the C level, to continue to create and initialize `file` instances by calling `PyFile_Init()` after allocating the instance.

6.3 Import

For implementing the changes required for the `__import__` function to protect the `import` statement, the code implementing `__import__` was completely rewritten. As it currently stands within Python the `__import__` function is implemented entirely in C. While this is not bad in and of itself, it does raise the level of complexity for working with the code. In order to introduce the required flexibility into the implementation of `__import__`, two design decisions were made: to re-implement `__import__` in Python and to use importers and loaders as defined in Python Enhancement Proposal (PEP) 302 as much as possible [34].

The decision to re-implement `__import__` in pure Python code was a practical decision. There was no specific requirement that we not work with the existing C implementation. It was decided that it would be more expedient to rewrite the function than trying to work with the existing C code in order to make the second design decision we made easier.

Our second design decision to follow PEP 302 is for flexibility reasons. PEP 302 specifies a system that allows for flexible control over how modules are imported. PEP 302 builds off of Python's idea of a search path, much like the `PATH` environment variable on UNIX operating systems. Stored at `sys.path`, it is a list of strings specifying places for the import machinery to look for modules.

PEP 302 extended this simple approach in two keys ways. One is the introduction of an API for importers and loaders. Importers are objects that implement an interface to be queried as to whether a path entry can handle the requested import. Loaders are objects that handle the actual loading of the module. There is machinery in place to associate each entry on `sys.path` with an importer so that each location can be handled individually. Implementing `__import__` in terms of PEP 302 in this regard means separating the aspects of finding and loading modules into objects for the various kinds of modules that Python can import. This provides better modularity and control over how different types of modules are handled. It is also different from the original state of affairs where a monolithic function call at the C level handled the importation of modules.

The other key way PEP 302 enhances how Python imports modules is the introduction of a meta path. While `sys.path` works for modules that have a filesystem location, not all modules have a specific location. Modules that are built into Python do not have a sense of location, for instance. The addition of `sys.meta_path` provides a path similar to `sys.path` such that modules that have no filesystem location can be included in the search path for modules.

With an implementation of `__import__` that follows PEP 302 the needed level of modularity is available to whitelist modules based on the kind of the module being requested. An importer and loader for built-in modules and frozen modules has been written that resides in `sys.meta_path`. For pure Python source code, bytecode, and C extension modules, an importer and loader that can be configured to handle any of the three mentioned types of modules has also been written that handles path entries on `sys.path`. All importers and loaders can either be left out of the system so as to completely block the importation of a specific kind of module (as with Python bytecode modules) or to place a whitelist delegate in front of an importer and loader (as for built-in, frozen, and C extension modules).

But how does one prevent the manipulation of the whitelist if Python lacks any form of private namespace for objects? To handle this a two-part solution is used. First, the object called to handle imports is stored in the `sys` module in the `import_` attribute. This can be safely done as the `sys` module is not exposed by default as it is a built-in module. As long as a user does not whitelist `sys` then code running in the interpreter cannot reach the object stored at `sys.import_`.

Second, a simple delegate is implemented in C that when called passes its arguments to the object stored at `sys.import_`. By implementing the delegate in C, `sys.import_` is in no way exposed as discussed in Section 5.4.

One issue that came up during implementation is how best to handle modules that are cached during startup of the interpreter. The Python interpreter stores all imported modules in a Python dictionary stored at `sys.modules`. The cache allows future imports to return quickly from the cache. It also provides consistency by having all modules use the same instance of a module instead of a newly created instance for each import. Because the cache is kept in a module that is blocked from being imported by default it is not a security risk itself. The issue, though, is how to not expose modules that must be imported for the interpreter to function, and thus cannot be deleted, but should not be exposed by default (e.g., the `sys` module).

The solution used is to hide all modules that are required for the interpreter to function but are deemed dangerous to expose by default to be put into a Python dictionary that is stored in `sys.modules` under the `.hidden` key. The `__import__` function is changed such that it does not ever allow the importation of a module that has a leading ".". By hiding modules based on name it allows returning modules from the cache to be no more expensive than a simple string prefix check on the module that is being imported.

Placing a capability on all modules that are safely imported and checking for that capability when returning from the cache was considered as a solution to this problem. The current solution was chosen instead however, because it allows for providing an alternative implementation of a module that is hidden. If the capability approach was used then whether a module with a specific name could ever be imported would be an all-or-nothing proposition because name clashes would prevent an alternative module. With the selected solution that is not the case as the name of the module being hidden is not directly used in the cache.

7. Current Status

At the time of writing the implementation is not complete [1]. Both the `file` and `code` types have been modified. The `__import__` function has been rewritten in Python source code. The modules required for Python to run but should not be exposed have been moved into the `.hidden` key in `sys.modules`. Proper preven-

tion of directly trying to import `.hidden` has been implemented. `sys.modules` is also cleared of all unessential modules.

What has not been implemented is a proof-of-concept application where Python is used as a DSL. Because this has not been done some changes cannot be made on the testing interpreter as the build toolchain for Python includes a Python script that needs access to some functionality that is considered a security risk but runs with the newly built interpreter. The `open` and `execfile` have not been removed because of this build dependency. Whitelisting of modules has been implemented in the `__import__` function but it has not been integrated into the interpreter. Once the proof-of-concept application is written it is trivial to remove the mentioned built-in functions and integrate the whitelisting.

In terms of vetting the design presented in this paper, it has been presented in three separate venues. The first is the e-lang mailing list [2] where various people involved in the object-capabilities world discuss object-capabilities languages. The Python development team [4] is the second venue this work has been presented. Finally, at the PyCon 2007 Python community conference [12] a talk was given on the work. In all three venues no objection to the design was brought up and it received favourable reviews.

8. Future Work

Two directions where this work can go is to is to introduce a private namespace within the Python language itself and trying to extend the work to act as a replacement for the `rexec` module. In terms of introducing a private namespace for Python, it would remove the necessity to utilize the barrier between Python and C exclusively to implement delegates.

One possible way to provide a private namespace is through closures. Python contains support for closures where free variables can be accessed in a read-only manner (Python 2.6 will allow for read-write access to free variables). But closures cannot be used as a private namespace as they are currently implemented because the values of free variables are exposed through introspection (as of Python 2.5). If the introspection was removed then closures would be usable for delegates when written properly. Figure 1 has an example implementation of how removing introspection abilities on free variables could allow a security delegate for `open` to be safely implemented in Python code.

For re-implementing the `rexec` module, one possible approach is to utilize Python's ability to have multiple interpreters in a single process. By providing a module that allowed for executing code in a separate interpreter it may be possible to replicate the `rexec` module's abilities. It would require analysing what objects are shared between interpreters and making sure there are no covert channels. Analysis would also be needed to see how returned objects from the code executed in a restricted interpreter cannot gain escalated rights in the creating interpreter.

9. Conclusion

This paper has presented a design for a mechanism to restrict access to resources by code running within a Python interpreter. Thanks to the moduarity of the Python interpreter and the use of object-capabilities, no changes of the language is necessary to allow a certain level of security for Python code when using a single interpreter. Only minor changes or removal of key functions, methods, and types in the built-in namespace is needed that are directly viewable by running code. And with a redesign of the import machinery, proper protections are complete for preventing access to things such as files and sockets.

Several things could have prevented this solution from working. If the built-in namespace was not modifiable then any dangerous built-ins would have ruined any security. Had Python not used an exposed function for the `import` keyword then proper protections of external code would not have been possible.

And object-capabilities played a key role as well. Its design allows for using the system in such a way that the Python language did not require modification. By harnessing various parts of how Python's interpreter is implemented the required semantics of object-capabilities could be introduced into Python through the interpreter and not through the language.

```
from os.path import join, normpath

def protected_open(open_fxn, restricted_dir,
                   path_join, path_resolver):
    '''Return a closure that restricts the use of
    files to a specific directory.

    Parameters:
    * open_fxn
        Use to open files.
    * restricted_dir
        Directory that files are to be restricted to.
    * path_join
        Function that joins parts of a path together.
    * path_resolver
        *resolves a path (e.g., handles '..').

    '''
    # Define a closure.
    def open(path, flags='r'):
        '''Open a file at 'path' using optional
        'flags'.'''
        # Construct an absolute path.
        joined_path = path_join(restricted_dir, path)
        abs_path = path_resolver(joined_path)
        # Verify the path does not violate the
        # directory restriction that is in place.
        if not abs_path.startswith(restricted_dir):
            msg = ('path leads outside of allowed '
                   'directory')
            raise ValueError(msg)
        # Return the file object.
        return open_fxn(abs_path, flags)

    # Return the closure.
    return open

# In Python 2.5, the value of the 'open_fxn' free
# variable for the open_delegate function can be
# accessed at
# 'open_delegate.func_closure[0].cell_contents' which
# prevents closures to be used as delegates.
open_delegate = protected_open(open, '/tmp/restricted',
                               join, normpath)

# Prevent functions from being exposed through
# 'open_delegate.func_globals'.
del join, normpath
```

Figure 1. Example implementation of a security delegate for opening files. Requires that introspection on free variables of closures is not allowed.

References

[1] bcannon-objcap branch. Subversion code repository @ `svn.python.org`.

[2] e-lang mailing list. `http://www.eros-os.org/mailman/listinfo/e-lang`.

[3] Javascript security in communicator 4.x. `http://web.archive.org/web/20040621232800/http://developer.netscape.com/docs/manuals/communicator/jssec/index.htm`.

[4] Python development team. `http://www.python.org/dev/`.

[5] Python programming language. `http://www.python.org/`.

[6] Pythonology. `http://www.pythonology.com/`.

[7] `rexec` module. `http://www.python.org/doc/2.5/lib/module-rexec.html`.

[8] Java security overview, April 2005.

[9] ANUPAM, V., KRISTOL, D. M., AND MAYER, A. J. A user's and programmer's view of the new javascript security model. In *USENIX Symposium on Internet Technologies and Systems* (1999).

[10] BEATTIE, M. *Safe perldoc page*, perl 5.8.6 ed. Perl Foundation.

[11] BISHOP, M. *Introduction to Computer Security*. Addison-Wesley Professional, 2004.

[12] CANNON, B. Securing python. PyCon talk, February 2007.

[13] DEAN, D., FELTEN, E. W., WALLACH, D. S., AND BALFANZ, D. Java security: Web browsers and beyond. Tech. Rep. TR-566-97, Princeton University, February 1997; 20 Pages.

[14] GONG, L., MUELLER, M., PRAFULLCHANDRA, H., AND SCHEMERS, R. Going beyond the sandbox: An overview of the new security architecture in the Java Development Kit 1.2. In *USENIX Symposium on Internet Technologies and Systems* (Monterey, CA, 1997), pp. 103–112.

[15] KAMP, P.-H., AND WATSON, R. Building systems to be shared, securely. *Queue 2*, 5 (2004), 42–51.

[16] KAMP, P. H., AND WATSON, R. N. M. Jails: Confining the omnipotent root. In *In Proceedings of the 2nd International SANE Conference* (2000).

[17] KOVED, L., NADALIN, A. J., NEAL, D., AND LAWSON, T. The evolution of Java security. *IBM Systems Journal 37*, 3 (???? 1998), 349–364.

[18] LAURIE, B. CaPerl. `http://caperl.links.org/`.

[19] MCKUSICK, K. The jail facility in FreeBSD 5.2. *login 29*, 4 (Aug. 2004).

[20] MILLER, M., AND SHAPIRO, J. Paradigm regained: Abstraction mechanism for access control, 2003.

[21] MILLER, M., YEE, K., AND SHAPIRO, J. Capability myths demolished, 2003.

[22] MILLER, M. S. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*. PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA, May 2006.

[23] OUSTERHOUT, J. K., LEVY, J. Y., AND WELCH, B. B. The Safe-Tcl security model. *Lecture Notes in Computer Science 1419* (1998), 217–??

[24] PAUL, N., AND EVANS, D. .NET security: Lessons learned and missed from java. In *Proceedings of the 20th Annual Computer Security Applications* (2004).

[25] PERL FOUNDATION. *Perl security perldoc page*, perl 5.8.6 ed.

[26] PETERS, T. PEP 20: The zen of python.

[27] RANUM, D., MILLER, B., ZELLE, J., AND GUZDIAL, M. Successful approaches to teaching introductory computer science courses with python. *SIGCSE Bull. 38*, 1 (2006), 396–397.

[28] SALTZER, J. H., AND SCHROEDER, M. D. The protection of information in computer systems. *Communications of the ACM 17*, 7 (July 1974).

[29] STIEGLER, M., AND MILLER, M. A capability based client: The darpabrowser. Tech. Rep. BAA-00-06-SNK, Combex, June 2002.

[30] THOMAS, D., FOWLER, C., AND HUNT, A. *Programming Ruby: The Pragmatic Programmers' Guide, Second Edition*. Pragmatic Bookshelf, October 2004.

[31] VAN ROSSUM, G. *Extending and Embedding the Python Interpreter*. Python Software Foundation.

[32] VAN ROSSUM, G. Python language reference. `http://docs.python.org/ref/ref.html`.

[33] VAN ROSSUM, G. Bastion too (was: Cross compiling). python-dev email, January 2003.

[34] VAN ROSSUM, J., AND MOORE, P. PEP 302: New import hooks.

[35] WHITTEN, A., AND TYGAR, J. Usability of security: A case study, 1998.

[36] ZDANCEWIC, S. Challenges of information-flow security. In *Proceedings of the 1st International Workshop on the Programming Language Interference and Dependence (PLID'04)* (2004).

Speculative Authorization

Jeremy Hilliker
UBC EECE 512 Project Report

Introduction

In authorization systems, a principal requests an action to be preformed. A policy enforcement point (PEP) receives this request and must enforce a policy decision made by a policy decision point (PDP). The PEP makes an authorization policy request to PDP, which decides if the actor should be granted authorization to preform the action.

The PDP may be a module within the same program as the PEP, the PDP may be in a separate process, on a separate host, or it may reside on a separate network. The PDP may have to query other data sources to retrieve information needed to make the policy decision. It may need to know human resources information, group memberships, site-specific information, or other data for business rules.

The query made to the PDP, processing done by the PDP, and the time that the PDP waits to receive responses to its own queries are a source of latency in the system. The PEP (and by extension, the principal) must wait for the query to the PDP to finish before they can proceed with their work.

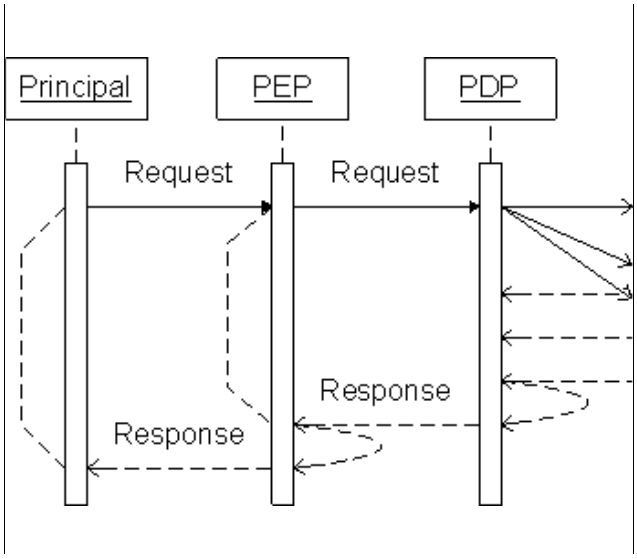

Figure 1 Authorization call diagram

Purpose

The purpose of this project was to propose and investigate a method for reducing the latency between when an actor requests an action and when the PEP can enforce a policy decision on that request.

Significance

In simple cases, the authorization latency is not a major source of problems. In distributed systems, and in real-time systems, the latency for an authorization can be a non-trivial delay.

Method

To reduce this policy delay, we speculatively pre-decided potential requests. The authorization system develops a model to predict what the actor's next request will be based on all actors' previous requests. Given all actors' previous requests, and this actor's current request, we predict what the actor's next likely request will be.

$$r^{*}_{1:k-1}, r^{i}_{k} \rightarrow f\left(\cdot, \Theta\right) \rightarrow r^{i}_{k+1}$$

The prediction of the next request is used to speculatively decide (to compute "permit" or "deny") the predicted action. The PEP caches this speculative result for potential use in the future, thereby bypassing the PEP to PDP delay in the principal's next request if the prediction was accurate.

Though a raw prediction of what may happen next is useful, it is more beneficial to attach a degree of certainty to the prediction. To this end, we predict the probability of the principal's next request based on previous requests.

$$P\left(r^{i}_{k+1} \middle| r^{i}_{k}, r^{*}_{1:k-1}\right)$$

Computing the prediction's probability allows the prediction algorithm to be tuned for performance. If prediction misses are costly, or if there are idle resources, a more aggressive system could speculatively authorize requests of lower probability. If resources are scarce, or if mis-predictions are expensive, then a conservative algorithm may choose to only execute predictions that have a high probability of use.

The prediction is also used to determine the cache policy. Since we compute the probability of the next potential request (along with request itself) we already have an indicator of the likelihood of that request being needed. It is not necessary to hold old predictions since the model would have re-predicted them if they were probable.

Given this framework, we can construct a system that speculatively authorises requests for principals based on their previous requests.

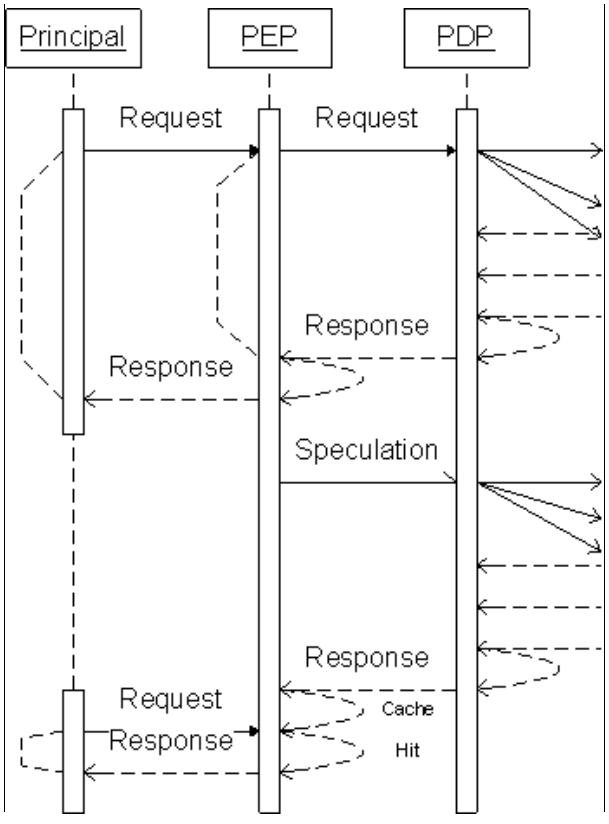

Figure 2 Authorization with speculation

Assumptions

The proposed solution to the problem depends on the following assumptions:

- There is authorization latency: there is some latency to be avoided by the speculative authorization. This solution would not be useful if there was no appreciable authorization latency between the PEP and PDP to be eliminated.
- There are idle resources which can be used to speculatively authorize requests: the computing of speculative authorizations would not negatively effect the system's throughput.
- Authorizations have a usable shelf-life and can be cached. This solution is not useful if authorizations must be determined based on real-time parameters that cannot be pre-computed.

Approach

The system is modelled in a data-driven discrete event simulation (DES) as follows:

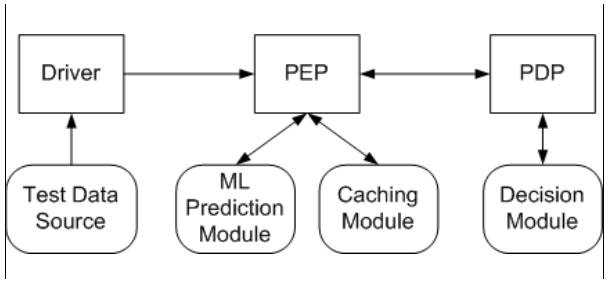

Figure 3 Simulation framework

Data Source

In the absence of a real-world trace of authorization requests from multiple users over diverse resources, a log file from a campus web server was used to simulate requests. This kind of data-set has been studied for prediction in [2,4,5].

The previous studies have noted how noisy the data is, and have proposed methods to scrub the data to make it more useful. Hansen (in [2]) proposes the following techniques to clean web logs:

- Data reduction: only consider requests for documents (HTML, TXT, PS, and PDF files.)

- Grouping and cleaning: establish series of requests based on requester IP. Use this to eliminate requests by robots, and try to eliminate requests made from shared IPs.

Additionally, any request made by clients with a "user agent" string that looked like a robot was ignored, requests to identifiable scripts (those under "/cgi-bin/" or where the URL contained a "?") were ignored, and request methods other than "get," "post," or "head" were ignored.

The simulation was run with 10 consecutive days of log data. The logs totalled 1.8 GB in size, and contained over 8 million records.

Sessions were considered to be ended after 20 minutes of inactivity. After this time, cached authorizations were invalidated, a subsequent request from a principle was considered to be a "first" request for a new session, and therefore would not be counted as a cache miss.

After cleaning, the log file consisted of only 230,000 predictable entries (entries which passed the filtering criteria, and were not a "first" request.)

Caching

Each principle was given an associated cache to hold their predicted authorization results. The cache size was unbounded, but effectively limited by a specified prediction threshold.

The system speculatively authorizes any predicted request that has a prediction probability above a specified threshold. For example, if a prediction threshold of 30% is given and the model predicts request A with 40% certainty, request B with 35% certainty, and request C with 25% certainty, then both requests A and B would be speculatively determined and cached.

When given a threshold of 33%, the cache could only hold a maximum of three entries per principal. 34% would give an upper bound of two entries, and 51% would yield one. The cache was also limited through the expiration of results after 20 minutes.

Markov Models

A Markov model is a stochastic process which is used to predict state transitions in a discreet space, meaning that when given a state, the model will generate a probability distribution for the next state.

Formally, a Markov chain is a sequence of random variables with the property that the future state depends only on the present state, and is independent from the past states.

$$P\left(X_{k+1}\middle|X_{1:k}\right) = P\left(X_{k+1}\middle|X_k\right)$$

A Markov model is said to be *n*-th order when this restriction is reduced to allow the last *n* states to contribute to the probability of the next state. A 2nd order Markov model would be of the form:

$$P\left(X_{k+1}\middle|X_{1:k}\right) = P\left(X_{k+1}\middle|X_k, X_{k-1}\right)$$

Higher order Markov chains have the advantage of having more data to work with, and may therefore make better predictions, but they have the disadvantage that they need more data to train from. A higher order model will need more data to fill its cells as the number of cells increases exponentially with respect to the order of the model. A first order model has n^2 cells, a second order model has n^3 cells and so on.

Markov Models as Predictors

If we consider the range of possible requests from principals to be discrete and finite, we can model the principals' requests as state transitions within a Markov process. The probability distribution can be represented by a stochastic matrix (called the transition matrix T).

One dimension of the matrix represents a previous requests, and the other represents the subsequent request. Each cell will indicate the number of times that transition has occurred. As such, the probability of one request following another is:

$$P\left(x\middle|prev\right) = T\left(prev, x\right) \Big/ \sum_{next} T(prev, next)$$

Additional dimensions can be added to the matrix to model higher order Markov processes.

In our simulation, the Markov model starts empty and is populated as each request is made. If the first request is for *A*, and the same principal's second request is *B*, then the entry *T(A,B) = 1*. A request for *A* from a second principal will yield a prediction of *B* as the next request with 100% probability. If the second principal then requests resource *B*, the table is updated as *T(A,B) = 2*. If instead the second principal requests resource *C* then the table is updated with *T(A,C) = 1*. In the latter case, if a third principle makes the next request for resource *A*, then its predicted next access will be for *B* and *C*, each with 50% probability. This way, the model will learn in an iterative fashion, make more and more accurate predictions over time.

Results

The simulation was run with Markov models of the first, second, third and fourth order, each with prediction thresholds of 30%, 40%, and 50%.

Number of cache hits, misses and total predictions made were tracked as well as the execution time for each run.

The table below summarizes these results. Hits, misses, and predictions are given relative to the total number of predictable requests, and execution time is given relative to the baseline of no predictions.

Predictor	Threshold (%)	Hits (%)	Misses (%)	Predictions (%)	Time (%)
None	N/A	0	100	0	100.00
1st Order Markov	30	44	55	126	103.73
	40	40	59	96	101.66
	50	38	61	82	101.45
2nd Order Markov	30	50	49	136	101.42
	40	42	57	97	100.76
	50	40	59	83	102.16
3rd Order Markov	30	50	49	130	100.94
	40	44	55	99	100.79
	50	40	59	83	102.66
4th Order Markov	30	48	51	126	102.65
	40	43	56	89	102.39
	50	39	60	81	104.41

Discussion

Fifty percent prediction accuracy was achieved in the best case, and this is believed to be an indication that the algorithms work well.

Access patterns to a website are anecdotally very haphazard. Consider a front landing page with a dozen or so links to various sub-pages. One would expect that each of these sub-pages are visited with relatively equal frequency, and that making predictions would be hard. This model was able to predict the top three next-page requests with 50% certainty, and the top one and two next requests with 40% accuracy.

As shown in the table, the prediction threshold is the largest factor affecting the performance of the system. Lower thresholds are more aggressive, and will generate more hits and fewer misses. This parameter can be changed to trade the time and space used for speculation and caching against the time saved by cache hits. The system could dynamically adjust this parameter to maximize the amount of predictions that it can make wile staying within some specified resource limits.

The second order model is noticeably better than the first order model, the third order model is only marginally better than the second order, but the fourth order model is slightly worse than the third order model.

The decrease in performance in the fourth order model can be attributed to not having enough data to train from. The fourth order model is very large, and when not given a sufficiently large data-set, it become sparse and its predictions are over-fitted (it trains too closely to match its input at the expense of future accuracy).

Each increase in model order raises the space requirements by an exponential factor, so the first order model should be a sufficient predictor in most cases. If space requirements are not a major concern, then the second order model can give some improved prediction performance.

The execution time required by the prediction algorithms is very small when compared to the total execution time of the system. In our simulation, the total execution time is dominated by the parsing of the log file (using a widely used regular expression library). In the worst case, the model only accounted for 4% of the execution time

Future Work

The performance of the first order model is nearly as good as the second order model – so much so that the increased space requirements for the second order model do not seem justified.

The increased input parameters of the higher order models give those models more information to work with to make their predictions. They essentially use the longer access history to predict what kind of user the principal is, and tailor the predictions to that type.

It is possible to capture and model these principal groupings in a more condensed way.

If we can determine clusters of users, we can train different models for each of those clusters. The clusters will predict the type of principal, and the model for that cluster will be an expert predictor for that type of user. These experts predictions will be mixed together based on the certainty of the clustering, and will produce a weighted mixture of the predictions.

More specifically, we hope to predict the probability of the actor's next request based on their previous requests, and on a marginalization over a learned clustering of users.

$$P\left(r^i_{k+1} \middle| r^*_{1:k}, i\right) = \sum_g P\left(g \middle| r^i_{1:n}\right) \cdot P\left(r^i_{k+1} \middle| r^*_{1:k}, g\right)$$

This has the potential to realize the gains of the higher order Markov model while maintaining the same exponential space requirements.

Mixture of Experts

In this approach, several predictors are created for clusterings of principals, and these predictors are weighted against each other based on a marginalization over the probabilities that a principal is within that cluster. The predictors the clusters, and the number of clusters can be learned from the data.

Both [2] and [4] have used the EM algorithm [3] for clustering, and state that first order Markov models are suitable for predicting web access when used in a mixture.

EM is an iterative algorithm which assigns a probabilistic determination that a given point belongs to each cluster (the expectation step). The next iteration that moves the mean of the clusters and calculates the covariance for that mean along each feature dimension (the maximization step). These two steps are repeated until the means converge to some point within a given convergence threshold.

Although the referenced literature has found this method to be the most accurate with acceptable performance characteristics, it has only been applied to analysing static data. This method has not been examined with consideration to predicting live data where the model must retrain as it is in operation.

Conclusion

We have shown how a Markov model can be used to predict what a next principal's request will be, and we have shown how to train that model in real-time using a live data-stream.

The performance of the prediction algorithms were excellent, accounting for less than 4% of total execution time, and yielding 50% accuracy.

References

1. Beznosov, K. (2005). "Flooding and recycling authorizations." In Proceedings of the 2005 Workshop on New Security Paradigms (Lake Arrowhead, California, September 20 - 23, 2005). NSPW '05. ACM Press, New York, NY, 67-72. DOI= http://doi.acm.org/10.1145/1146269.1146285
2. Hansen, M.H. and Sen, R. (2003). "Predicting Web User's next access based on log data." Journal of Computational and Graphical Statistics, 12(1), 143-155.
3. Bilmes, J. (1998). "A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models." Technical Report TR-97-021, Int'l Computer Science Inst., Berkeley, Calif.
4. Cadez, I., Heckerman, D., Meek, C., Smyth, P., and White, S. (2003). Model-Based Clustering and Visualization of Navigation Patterns on a Web Site. Data Min. Knowl. Discov. 7, 4 (Oct. 2003), 399-424. DOI= http://dx.doi.org/10.1023/A:1024992613384
5. Huberman, B. A., P. L. T. Pirolli, J. E. Pitkow, and R. M. Lukose (1998), "Strong Regularities in World Wide Web Surfing," Science 280, 95--97.
6. Bishop, C. (2006). Pattern Recognition and Machine Learning. Springer.

People-Tagging: Towards Usable Privacy for Social Software

Maryam N. Razavi
University of British Columbia
2366 Main Mall, Vancouver, BC
1-604-827-5909
Maryamr@ece.ubc.ca

ABSTRACT
As the use of social software for various personal and professional purposes gets widespread, the issue of providing usable support for managing access to the vast amount of user-created content in such an open environment becomes more of a concern. In a recent work, we proposed a grounded theory of how users manage privacy of their information in a typical social software where information sharing and online collaboration is encouraged and users are producers as well as consumers of information. The grounded theory suggests that users' preferences regarding privacy of their artifacts in such an environment depends on a number of factors, including the current stage in the artifacts life cycle, the nature of trust between the owner and the receiver of information, and the dynamics of the group or community within which the information is being shared. In this paper we discuss what the results of the theory mean for designing more usable privacy management mechanisms for social software, and why existing access control models are insufficient in supporting specific privacy requirements in this particular context. Based on our findings, we then present the design and implementation of a privacy control mechanism based on tagging people that provides a more usable privacy management mechanism for social software.

Categories and Subject Descriptors
H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces – *Theory and model;* H.1.2 [**Models and Principles**]: User/Machine Systems – *Software Psychology.*

General Terms
Human Factors, Theory, Design, Security.

Keywords
Social software, grounded theory, information sharing, information privacy, tagging.

1. INTRODUCTION
Recent advances in the emergence and growth of Web 2.0 applications have made the users producers as well as consumers of information. Social software is a by-product of the Web 2.0 phenomenon: the second generation of internet-based services that include some form of many-to-many publishing (such as social networking, social book marking, weblogs, wikis, and ePortfolios), enhanced organization and categorization of content, and most importantly, encourage generation and distribution of Web content. The Web 2.0 phenomenon is characterized by open communication, decentralization of authority, and freedom to share and re-use [12]. Although sharing information is one of the major motivations behind the use of social software, not everything is to be shared with everyone. While use of social software has moved within the reach of non-technical mainstream, managing selective sharing of published information still requires expertise. Lack of proper access management mechanisms has been identified as one of the major impediments in the wide spread use of social software despite its obvious benefits [22, 3]. Research into access management has generally concentrated on the needs of organizations or distributed systems. However, privacy requirements for user-created content in social software are different from data protection requirements in organizational databases and operating systems: social software is often used for both social activity and engaged work practices and as such, provides users with the opportunity to include a wide variety of artifacts in their environment, from work related documents, to personal opinions expressed in a weblog, to bookmarks and personal collections. Over time, this aggregation of ones' information could present a rich view of his experiences and skills in the form of a searchable life log. This creates a persistent, long-lived online identity for the user, to which he may wish to expose different views to various audiences. The shared artifact and the groups in which it is shared could both be dynamic, and preferences regarding sharing the artifact within a group have to be flexible enough to accommodate frequent changes. Information is not necessarily shared with identifiable, accountable individuals, and sharing might happen in various contexts, for example competitive as well as collaborative.

Traditional access control models generally address the problem of enforcing well-defined rules set by central authorities and do

not account for the dynamic nature of personal preferences as required by social software. On the other hand, access control models that are proposed in the literature for groupware (e.g. [31, 6]) tend to be rather complex and leave the important question unanswered whether users will be able to cope with this complexity. Thus, there is a clear need for privacy management models that address specific privacy requirements in social software and yet, are easy enough for non-technical users to understand and use. Our research has been motivated by this need: we believe that in order to be usable, privacy mechanisms must reflect users' needs and must be built based on users' mental model of information privacy. To this end, we recently conducted a grounded theory study of users' information sharing behavior in social software to identify users' privacy needs. The study showed that users' privacy preferences depend on a number of factors, including the current stage in the artifacts life cycle, the nature of trust between the owner and the receiver of information, and the dynamics of the group or community within which the information is being shared. In this paper we discuss how the results of the theory can be translated into guidelines that inform design of more usable privacy management mechanisms for social software. We also discuss some of the existing access control models and their insufficiencies in supporting specific privacy requirements in this particular context. Based on our findings, we then propose a privacy control system to provide more usable privacy management for social software.

2. RELATED WORK

In recent years, use of social software has moved from niche phenomenon to mass adoption [10, 22]. This increase in use has been accompanied by diversity of purposes and access patterns. As a result, researchers have studied several issues that pertain to these tools, including people's attitudes towards disclosing personal data.

Gross et. al [10] report on a study on patterns of information revelation in online social networks and their privacy implications. Their results are based on actual field data from more than 4000 users of Facebook. They report that patterns of information revelation depend on a number of factors, including pretense of identifiability, type of information revealed or elicited, and the degree of information visibility.

Researchers have also studies users' attitude towards revealing information in several other contexts, including work place, online services, and location-aware mobile services. Olson et. al. [26] take a quantitative approach in conducting an in-depth survey of people's willingness to share a range of everyday information (such as web sites they visit or their health status) with various others, including family members or co-workers. They point out that whether data is anonymized or can be tied directly to people plays a major role in people's willingness to disclose. Other relevant factors reported include general attitude towards privacy (from privacy unconcerned, to privacy pragmatist, to privacy fundamentalist), and personal judgment regarding "appropriateness" (i.e. relevance) of sharing certain information with certain groups.

In another work, Patil et. al [27] conduct a study on privacy/awareness tradeoff to identify the kinds of information that users of an awareness application are willing to share with various others (team mates, family, friends, managers, etc.) for various purposes in the context of the workplace. They identify which clusters of awareness information are more likely to be shared with whom and in what context (i.e. "team members" received comparable level of awareness sharing with "family" during work hours).

Whalen and Gates [35] report on a small-scale study on the type of personal information that users would be willing to disclose in open online environments, primarily focusing on uncontrolled spaces such as search engines. Their results, although limited in scope, point to the existence of consistencies in the way people treat certain classes of information, which suggests it might be possible to group related information into clusters that are treated similarly.

Recent works in the area of knowledge management (KM) have also recognized the need to improve people's ability to control who sees what in their personal information. Erickson [7] explores the concept of personal information management in group context, by arguing that when personal information is to be shared with a group, the way it is used and managed changes. In his article on GIM, Group Information Management, he identifies many research questions that need to be explored, including how personal information is shared within a networked group, the norms of personal information sharing within groups, and the way those norms are negotiated in the group.

3. THE STUDY

In the view of the difficulties that HCI researchers have encountered in locating places where the context of privacy can be better understood, we undertook a qualitative study with the aim of identifying privacy needs, concerns, and challenges in social software from users' perspective. The research method that was selected for the study was grounded theory [9, 23]; a primarily inductive investigation process in which the researcher aims to formulate a small-scale, focused theory that is derived from the continuous interplay between analysis and data collection. A comprehensive explanation of the study and the overall theory is provided in [28]. In this paper, we focus on the results of the study and their implications for design.

Two main themes emerged from our grounded theory study: First, we determined that privacy *is* a main concern of users of such systems, and second, we identified factors that affect users' privacy preferences. The next subsections present a more detailed description of each theme.

3.1 Centrality of Privacy as a Concern

The concept map in figure 1 highlights the centrality of privacy as a concern for our subjects. Although the tool was primarily introduced to participants for educational purposes, they were also using it for interacting with each other (social networking), writing personal reflections (weblogging), and showcasing samples of their creative works. This confirms that as with other computer-mediated social technologies (e.g. email [36]), when given a rich environment that provides support for both work related and social activities, user communities will adapt it for more purposes than was initially conceived. Many participants mentioned they see potential benefits in using the tool, such as having all their information in one central place and over the Internet, where they can refer potential audiences to view things rather than having to send them stuff individually. Many also mentioned that it helps them keep track and reflect on their improvement over time and in some cases, get unbiased feedback

on their creative artifacts from a community of people who share the same interest.

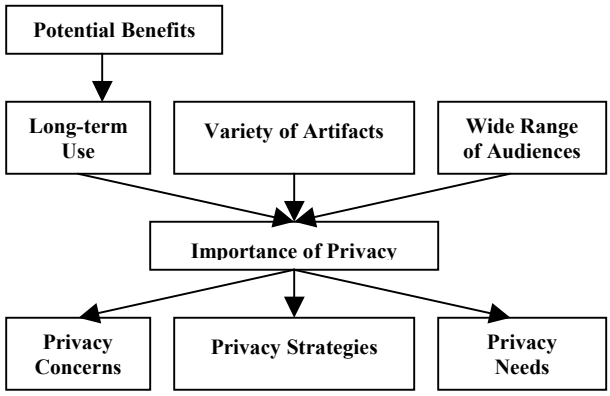

Figure 1. Centrality of privacy as a concern

However, most participants agreed that the tool did not provide sufficient support for privacy control for all their various needs: they had a wide variety of artifacts (ranging from personal profile and reflections to educational material and creative work) targeted to a different groups of audiences (teachers, classmates, friends, various communities) that were not necessarily static. These specific characteristics of the environment plus the tendency for long-term use, gave rise to a need for selective sharing. Two major concerns that were brought up by most users were the concern over loss of control and credit (mostly for creative and educational artifacts), and the concern that their work might be interpreted out of context (mostly for personal opinions and reflections). Because of these concerns, many of the participants employed certain strategies to achieve the desired level of privacy: Some were using other platforms with better privacy management mechanisms for their more private content; others had chosen to stick to one platform, but write their more private content in some sort of a "code language" so that it was meaningless to anyone other than the writer himself; and some had decided not to provide a link to certain material from places where their real identity is known. These strategies pointed to the fact that there are certain privacy needs of the users that the tool fails to support. Almost all participants mentioned that a better privacy management mechanism would improve their experience with the tool.

3.2 Privacy Factors

The second theme that emerged from our grounded theory were the factors that affect privacy of information from users' perspective as shown in figure 2. Our study showed that rather than a binary scale of public vs. private, users' judgments of privacy of resources often reflects a transition from private, to semi-private/restricted share, to public, depending on the state of the information, the receiver, and the context of sharing. The study showed that users have different perceptions of privacy of their artifacts in different stages of the artifact's life cycle. For example, an artifact is often considered private at the time of creation when descriptions, goals, and personal reflections are often included with the artifact. However, during the work-in-progress stage, the artifact may be shared with a restricted audience to obtain feedback, and it then may be shared with a larger/more public audience once it is completed.

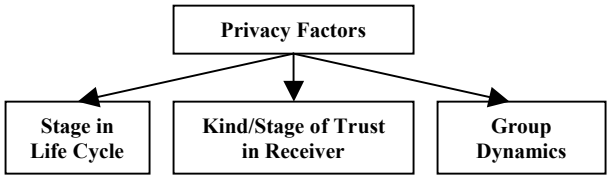

Figure 2. Factors affecting privacy preferences

We also found that users' assessment of the persons or groups who will be the receivers of information plays a strong role in making decisions about information sharing: users tend to share less with people/groups with whom they are in the initial stages of trust, and as their trust moves towards a more mature level over time, they begin to feel more comfortable and share more. The most influential factor in the information sharing attitudes of users however seems to be the dynamics of the groups or communities where the information may be shared. Our study revealed that users often hold back from sharing information in anticipation of loss of control and influence, and loss of credit for their work. The theory suggested that when group/community dynamics are clear enough to convey to the users how their information will be used within the group, users may be better equipped to make informed decisions regarding how much they want to share within the group. Moreover, this predictability may be critical to making the decision to share information in the given context at all.

4. TOWARDS A CONCEPTUAL MODEL OF INFORMATION PRIVACY

The main objective of a grounded theory study is to improve understanding of a phenomenon and to construct an evidence-based theoretical framework describing the phenomenon. In general, whether it is based on qualitative or quantitative evidence, a theory has both explanatory and predictive force. Whereas a theory may be initially accepted based on its explanatory force (especially if it is about something that is unexplained or insufficiently described), its perceived usefulness is determined by its predictive force. To that end theories often include a model (either formal or informal) that others can test and apply. As such, the model is expected to make predictions that can be evaluated in different situations. In this section, we extend the results of our grounded theory into a conceptual model of information privacy for social software. We discuss how the privacy factors that were identified by the theory can be translated into requirements that should be supported by social software systems, and then follow up by a discussion of existing access control models and their ability to support these requirements in the next section.

4.1 Analysis of Findings

The first observation that follows from our study is that users have nuanced ideas about what they want to share with whom and in what context, and they consider it a shortcoming of the tool when their desired level of privacy is not supported. In some cases privacy may even determine their choice of tool or their level of engagement with it. The fact that users try to address lack of

desired level of privacy with certain strategies points out to the fact that even though users might adapt their behavior to the tool, that does not mean the tool is good enough for their purpose, which further emphasizes the need for privacy management mechanisms with better support for users' needs.

Considering the three privacy factors that emerged from the second theme, we can see that the theory suggests that in practice, users view the information sharing act as establishing and maintaining a *dynamic* sharing relationship, rather than a single event. Although the information sharing act seems like a simple and straightforward act (user shares something with a group of receivers based on their current relationship), there are various levels of dynamics to this model. Over time the artifacts might change (i.e. research results getting published, patented ideas getting approval, personal opinions reconsidered), the receiver group might change (i.e. competitors joining a group or collaborators leaving), and the relationship between the user and the receiver group might change (i.e. switching to a different project, change of affiliation). In short, all the contributing factors in users' privacy preferences can change over time, and all the three factors that were identified by our study reflect users' needs for support of these kinds of changes: The *privacy life cycle* factor emphasizes the effect of the dynamic nature of the artifact; the *trust* factor reflects the effect of change in the relationship between the user and the receiver; and the *group* factor shows how users try to deal with these changes: organizing ones' network into various groups is a way of compartmentalizing trust and audience, rather than having to deal with it on an individual basis.

4.2 User-Oriented Privacy Controls

The central structure upon which we ground our design model is the description of the kinds of control of privacy that have been shown to be necessary for social software systems for managing and sharing personal information or work products. In short, the theory suggests that users need *artifact control*, *audience control*, *relationship control*, and most importantly, *change control* for all the three of the factors outlined above.

4.2.1 Artifact Control

The principle of *artifact control* reflects the need for control of the privacy of information in terms of both individual artifacts and collections thereof. This is of course mere confirmation of the long-standing model that access rights should be associated with individual objects (e.g. files) and their collections (e.g. folders). But since social software has a different granularity and object creation model, it may be that the way in which these rights are managed to protect privacy and facilitate sharing needs to be different in some essential ways.

Our study suggests that unlike static artifacts for which the set-on-creation access management models may be sufficient, the dynamic nature of the personal artifacts that are generally disposed in social software systems calls for more fluid rights management techniques. For dynamic artifacts, users seem to dynamically match privacy and control to the artifact's degree of completion. We believe we can use this fact to reduce the up-front cost of privacy management by gathering privacy context from the environment. Since users already categorize their information for other purposes, it makes sense then to leverage these categories further by associating default access patterns with different user-defined categories. Of course, categories can be defined in various contexts and tuned to the application. They could be established globally as a library of workflows that can be used by individuals or groups or built from the ground-up by the users and shared like other artifacts within the social software system. If we hope to provide a global library of such templates though, it will be necessary to align the models with preexisting mental models in order to guarantee out-of-the-box usability. We suggest that providing a set of such patterns that offer both static and dynamic rights management would help give control to the users without too much overhead: once a pattern is selected for an artifact, the access restriction level of the artifact can be changed by simply selecting which stage of its life cycle the artifact is currently in.

Furthermore, categories with default access patterns can help catering to the needs of both novice and expert users by conforming to the principle of safe staging [37]: users can choose to accept the defaults while they are in the initial stages of interacting with the tool, and once they have moved to a higher level of expertise, they might decide to modify or extend the defaults to better suit their needs.

Finally, if the categories are themselves treated as resources to be shared, discussed and managed then such an evolution may actually happen on a community-by-community basis.

4.2.2 Audience Control

The principle of *audience control* reflects the observation that from a user's point-of-view, the primary concern in assessing the information sharing act is in understanding the audience that will have access to that information. From an access control point of view, this suggests that the most significant access rights to be modeled are those pertaining to the mere visibility of an artifact (e.g. does it even exist at all) and its readability (i.e. ability to access its contents). For user-oriented privacy management, we will use the term "*audience control"* to describe the ability to restrict the visibility and readability of artifacts to certain user-defined groups.

We see some of this control currently being expressed in certain social software systems, notably Del.icio.us [15], Orkut [20], and Facebook [16]. Del.icio.us was originally completely open (i.e. anyone could see anyone else's complete set of bookmarks) but due to user demand and competition from other social bookmarking services (notable Ma.gnolia [18] and Bluedot [14]) it added the ability to mark bookmarks as "private" in the Spring of 2006. A private bookmark in del.icio.us is essentially invisible to anyone else but the user himself. In Facebook and Orkut, are services that are largely concerned with identity construction and maintenance [2]. The greater risk of exposing identity attributes to a worldwide audience has thus resulted in the deployment of a great deal of audience control for one's personal profile information. In essence, one can choose which of a variety of different categories of "friend" and "colleague" will be allowed to see any particular piece of identity attribute (e.g. phone number, address, AIM id) or posted content.

Audience control is clearly most directly related to the group and trust factors described above. In essence, the choice of audience for a particular artifact or personal attribute is primarily expressed in terms of a *group* of others who one *trusts* with that particular piece of information. While there are many models of trust in the literature, we do not depend on any one in particular. It is important to note, however, that our grounded theory clearly

delineates that the trust one has in a particular group with which one might share information depends critically on the model by which the membership in that group may change over time. We will revisit this issue when we discuss change control below.

4.2.3 Relationship Control

The principle of *relationship control* reflects the finding that many of our information sharing needs can be described in terms of the relationship that exists between the owner of the artifact and the person or group with whom the information may be shared. At first blush, this seems simple and obvious, but in terms of rights management it strongly implies that the potential audience for some artifacts or attributes is likely defined in the user's own terms, and not in terms of any organizational "roles" or groups. In other words, each and every user needs the ability to define "groups" of friends or collaborators in their own terms and then to be able to use this model of their relationships with others as the basis for audience control (at minimum).

Again, we look at Orkut and Facebook for examples. In Orkut, a user is able to define an audience for identity attributes in terms of his/her self-designated "friends" and a limited transitivity of that friendship network (e.g. I'll let my friends and any friends of my friends see my phone number). In Facebook, the relationship categories are much finer and reflect a variety of different kinds of relationship (e.g. we worked together on a project, we "hooked up"). The consequences are similar, however, in that I can then choose to allow access to particular posts or personal attributes based on these relationships, but without the transitivity of the Orkut model. Of course, Facebook also has more traditional "groups" that are formed by users explicitly joining them as well, but the audience for user attributes and personal posts is controlled completely in terms of the *relationship control* that the system allows.

Relationship control is clearly a manifestation of the need to define *trust* in terms of ego-centric *groups* of users, so both of these factors are essential. Less obvious, perhaps, is the way in which this interacts with the *privacy life cycle* of artifacts. In essence, it is very likely that the best match for the assignment of audience and other rights (e.g. modification rights) to an artifact through its life cycle are via these relationship groupings, and not via traditional "group" or "role" assignments. Whereas it certainly makes sense for an organization to align access rights to organizational roles, it makes little sense for a user to align privacy rights with those organizational roles. Given that an egocentric relationship model naturally aligns with patterns of trust and information sharing for personal information, it is essential that audience and other access control rights be assignable based on these user-controlled relationship models.

4.2.4 Change Control

The principle of *change control* is something of a cross-cutting concern within the other control patterns. In essence, with social software systems one must never forget that the artifacts, audience and relationships used to define privacy and sharing patterns are dynamic. In essence, our privacy and user interaction models must reflect an assumption that artifact life cycle and categorizations will change, that a user's requirements to share classes of artifacts with certain audiences will change, and that a user's relationships and trust patterns within those relationships will change, and that the whatever access rights are consequences of these models must change whenever they do.

This principle then strongly suggests that a model that assigns access rights based on these factors at the time of an artifact's creation or modification will be inadequate. In essence, the access rights must reflect changes in whatever models are used to fulfill the above principles dynamically and visibly. This may be implemented in many different ways, but it essentially demands that either the access control regime be based on the privacy model directly or that whatever rules connect the privacy model to the access control regime be dynamic and incremental, reacting to whatever changes are made to the social parameters that define the sharing model.

This may, of course, require some rule-based system to maintain this connection (e.g. [4]), but it is likely to require significant interaction with the social software's notification system as well. For example, if we follow the user interface "principle of least surprise" [13], then when a user (A) adds some other user (B) to a relationship category, both A and B should be notified in some way of the consequences of this change (e.g. user B now has access to a new collection of information). For the initiating user (A) such a notification (or at least ability to explore the consequences of this action before it is taken) can be critical to making the decision in the first place. For user B, the granting (or restricting) of rights to a body of information is an important piece of data to be able to assess their own relationship model.

5. Candidate Access Control Model

We now examine some of the existing access control models and discuss their suitability to be applied to social software. For each model, we use the principles and philosophy behind the model as the basis for our discussion on its ability to support the user-defined privacy controls as discussed in the previous section.

5.1 RBAC

We start with RBAC [30], as one of the widely accepted models for managing access control in the literature. RBAC was originally designed for controlling access to services and resources within organizations. The main characteristic of the RBAC model that makes it a suitable candidate for use within organizations is the ability to assign enterprise-specific access permissions to organizational roles rather than individuals. As such, the success of RBAC model depends on clear assignment of roles to users, and access rights to roles, by the system administrator (thus no user-oriented artifact control or audience control is supported). The effectiveness of the model is based on the underlying assumption that there are pre-defined roles and that the role/permission association changes less frequently than user/role association (thus assuming no user-oriented relationship control and change control). While this would be a valid assumption for the organizations and commercial applications, it is not necessarily true for social software: users of social software do not conform to an underlying organizational structure and personal information is not always shared with identifiable, accountable individual. Assigning appropriate roles to these users thus becomes an irrational and ad-hoc exercise.

Although our study suggests use of user-controlled group definition as a way of enabling users to specify their trusted audience, using roles for that matter as pertained to organizational structures would be counter-intuitive: in order to provide support for user-oriented controls, role definition and assignment need to be performed by non-technical users, as opposed to a system

administrators with deep technical knowledge. Considering the dynamic nature of user-created resources, audiences, relationships, and privacy references in social software as shown by our study, this calls for frequent changes in user/role assignment that needs to be handled frequently by the user, which would be too labor-intensive and counter to RBAC philosophy. We conclude then, that RBAC is not a suitable candidate for privacy management in social software.

5.2 TrustBAC

Over the years, researchers have proposed various extensions to the original RBAC model to tailor it to the specific needs of certain applications. One of these extended models is TrustBAC [5]. It adds the notion of trust levels to the original model. Users are assigned to trust levels instead of roles based on a number of factors like user credentials, user behavior history, user recommendation etc. Trust levels are then assigned to roles, which are then assigned to permissions as in the conventional RBAC. TrustBAC is proposed for open and decentralized multi-centric systems where the user population is dynamic and the identity of all users are not known in advance, such as service providers over the Internet.

In social software, however, access regulations to a large part depend on users' privacy preferences and attitude, rather than the receiver's credentials. Trust in social software mostly resembles the way face-to-face trust is shaped in the real world and between real people, which is based on implicit group norms and cultures rather than individuals' credentials. The notion of credential-based audience control as provided by TrustBAC with the addition of the notion of trust levels does not contribute to the support of any of the user-oriented privacy controls. Thus, like RBAC, TrustBAC is not a suitable candidate for privacy management in social software.

5.3 RelBAC

RelBAC [1] adds another level of abstraction to the original RBAC model by using the Resource Access Decision facility (RAD [29]) to include the notion of dynamic relationships between arbitrary entities in access decisions. The model is primarily targeted towards healthcare system, although the authors claim that it is general enough to be applied to any domain that requires relationships in access decisions. Relationships are explicitly defined using UML association or dynamic attributes. A combination of roles and relationships is then used to determine whether a permission should be granted or denied.

Like the original RBAC model, RelBAC is suitable in domains where there are clearly defined roles and relationships, for example when roles and relationships are dictated by requirements placed on access to information by the governmental or organizational rules. Considering the notion of relationships in addition to roles in making access decisions provides a more fine-grained right management compared to the original RBAC. The model supports relationship control (and audience control through it), but not in the user-oriented form as neither roles nor relationships are defined or controlled by the end users. Since the model is not geared towards dynamic information, the notion of user-oriented artifact control does not apply. Change control is indirectly supported through the assumption that relationships are short-lived and thus managed through other component of the system (e.g. registration component), rather than the central authority.

5.4 TBAC

The TBAC model is another extension of RBAC that introduces domains with task-based contextual information. Access control in this model is granted in steps that are related to the progress of tasks. Each step is associated with a protection state containing a set of permissions. The contents of this set change based on the task. This is similar to the concept of privacy life cycle as identified in our study, although permissions change based on various stages of tasks, not artifacts. TBAC is an active model that allows for dynamic management of permissions as tasks progress to completion. The model also supports validity period and expiration for the access rights.

The notion of artifact control is somehow supported because artifacts are assigned different permissions at different stages, although again this is not managed by the end users. Audience control and relationship control are handled through role assignment as in original RBAC and are not user-oriented. User-oriented change control is not supported.

5.5 BCSW

Sikkel [32, 33] presents a general authorization model with an emphasis on conceptual simplicity and ease of use. The model is provided in two forms: The basic form and the extended form. The basic form extends the canonical ACL model with the notion of groups that are used for both assigning roles (user groups) and permissions (access groups). The extended form adds support for delegation, negative rights (exclusion), conditional authorization, and explicit role switching. The model is modular in the sense that the extensions that are not needed in a particular application can be discarded, thus avoiding unnecessary complexity. Context (time, location, etc.) is also supported by the notion of conditional access rights applied to groups.

Use of *user groups* as the basic audience categorization mechanism (based on which roles and other kinds of categories can be modeled) seems to provide the level of flexibility in group definition as required in social software: Since user groups are collections of people without the attributes and operations for various types of roles, they enable group definition based on factors other than organizational roles. Compared to the notion of roles used in RBAC and other models that extend it, the notion of user groups in the BSCW model seems to be a better match for satisfying the audience control requirement in our conceptual model. Artifact control can be supported through the use of access groups, by assigning an artifact to different access groups through various stages of its life cycle. Also, the notion of conditional authorization in the extended model could be used as the basis for adding support for relationship control. Support for change control, however, depends on the actual implementation of the model and usability of the user interface that accompanies it.

As we can see, each of the discussed access control models is at best capable of *partially* supporting the requirements of social software, either directly or indirectly. Although it might be possible to modify or extend each model to *tailor* it to the requirements of social software, none of them are designed with specific needs of social software in mind. We now move to a short description of a privacy control system based on tagging people that we are incorporating into an open-source social networking

and information management system, and our plans to test this against other approaches. We claim that since our privacy system is designed with user-oriented privacy controls in mind, inherent support for specific privacy needs of social software is deeply integrated in implementation, potentially creating an intuitive, viable solution for usable privacy in social software.

6. Tagging people: a new model for relationship control

One of the most significant challenges in developing a system for audience and relationship control, and thus for supporting user-oriented privacy control and information sharing, is the subtle and nuanced way in which our patterns of trust change over time and the ways in which this interacts with our transactional approach to information sharing and exchange. In this respect, our study confirmed existing theories of knowledge sharing that compare the exchange of information between people with the exchange of money in economic systems [8]. While the analogy is not perfect, this observation highlights the contextual nature of the choice to share knowledge and the degree to which these choices depend on an assessment of the personal benefit to be gained from sharing weighed against the risk of sharing that specific information with that particular audience.

At minimum, a solution to this problem must involve an ability to associate collections of information-bearing *artifacts* with groups of people (the *audience*) defined largely in ego-centric terms (the *relationships*). The insight that leads to a potential solution is that the organization of relationships can be treated in the same way that we organize information itself, and that the model of personal information organization called *folksonomy* or *tagging* has exactly the characteristics necessary to facilitate relationship management for information sharing.

Simply put, the folksonomic information organization model allows a user to associate a set of personal keywords (tags) with a particular piece of information (an artifact). Each such keyword then automatically becomes a category term that can be used to select collections of artifacts for recall or comparison, using both individual keywords and certain Boolean combinations of these collections (as sets). Since this model was first introduced by the social bookmarking system del.icio.us [15] and the photo exchange system Flickr [17], it has been adapted to a wide variety of uses (e.g. blogging and RSS syndication), has become widespread in its exposure to the Internet community, and has been the subject of a body of research. While much of this research has been focused on the social aspects of the model, our interest is primarily on its usefulness as a model for organizing information for completely selfish purposes (what Vander Wal has termed *broad* folksonomy).

In relating information management to relationship management, we highlight a number of features of the tagging model:

1. Many tags (and thus categories) can be associated with each artifact;
2. The choice and control of tags is entirely in the control of the individual user;
3. The act of tagging is simple, intuitive and well-adapted to granular information collections (e.g. web bookmarks); and
4. The collections created by coincidental tagging (i.e. all artifacts tagged with the same words or the same set of words) form natural categories.

For these reasons, we propose to model relationships for information sharing by tagging people, represented by their profile pages in a social information sharing network.

Opntag is a web-based, open source system for note taking and bookmarking we have developed to experiment with personal information management and exchange in sensitive environments (e.g. within corporations). The fundamental unit of information storage in opntag is the "memo", a tagged textual annotation that may optionally refer to any URL-addressable object. Fundamental to its implementation is an ability to restrict the visibility of these memos to one or more groups (including the "private" group consisting only of oneself). To this point, we have used a fairly traditional model of user groups, based on the hierarchical BSCW model (e.g. a particular memo and its associated tags may only be visible to the "opntag developers" group).

One of the experimental focuses of opntag has been to exploit the opportunities presented by tagging or creating memos that refer to other objects in the system. For example, a memo that refers to another existing memo is considered to be a "reply" to that memo and becomes automatically threaded into the conversation that the first memo is part of. Memos on collections become associated with those collections and we are investigating the consequences of other tags applied to collections (e.g. in one experimental extension such tags are viewed as "implication markers" that signal semantic implication and automatic tagging).

Within this environment then we have started to experiment with the tagging of your own (identity tagging) and others profile pages as a way of "categorizing" friends and collaborators. This might be useful simply as a way of signaling our assessment of others (e.g. I might tag a seller on eBay as "unreliable") or as a way of signaling a relationship (e.g. I will tag my graduate students as "student" and "grad student"). When viewing that profile page then, I may be able to see the person's own tags for himself (e.g. a self-assessment of identity), my tags (signals of our relationship) and other's tags (third-party opinions). This is all, of course controlled by opntag's visibility management facility, so I will only see those tags that the taggers have allowed me to see, and thus it is reasonably safe to "opinion tag" others, but this is highly volatile and private information, so likely to be lightly shared.

As we have highlighted above though, associating the visibility of these tags with invitation-only or open membership groups (e.g. online communities) is probably not sufficient in most cases, since we often make such sharing decisions based on relationships more personal than shared membership in a community. For example, I may want my "friends" (i.e. those others I have tagged with "friend") to see that they are included and have special privileges to my information store as a result, but non-friends should not be visibly excluded. The obvious solution to this then is to treat each of these "tagged categories" of other users as a "relationship group" which is usable as a visibility category. Thus, the act of "tagging a person" (via their profile page) is equivalent to asserting their membership in a group *whose membership is entirely under my control*.

Currently, this implementation is incomplete and scientifically untested, but we can assert that it fulfills all of the control criteria outlined above. Sharing control within opntag is already done on

the basis of *artifact*-specific privacy control, since each memo in the system and its associated tags is visible only to its specific *audience*. The visibility management model in opntag is also clearly a user-driven *audience* control approach, with the audience for an item defined as the set of users the object is visible to[1]. The people tagging establishes the egocentric *relationship* control our study suggested and tying that to the visibility model allows one to exploit these relationships for audience management. The one aspect of the problem not directly addressed by this solution is *change control*, although the visibility of a memo or tag a user has created is always modifiable. What is needed is a way to match the changes in audience to identifiable stages in a privacy life cycle model, still to be developed.

The most salient comparison with this approach is the one exemplified by Facebook. A contrast to the bottom-up, user-defined vocabularies is the traditional application- or community-defined taxonomy. In Facebook, relationships are classified with a set of standard assertions represented by the dialogue in Figure 3: Facebook friend categories. We suggest that there are two problems with this model: 1) the categories are clearly incomplete (e.g. how do I indicate that I "taught" a student in a particular course?), and 2) I can't designate that individual photos, notes, etc. are to be shared with only a subset of my friends or networks (Facebook's groups).

Figure 3: Facebook friend categories

We are on track to complete the "tagging people" implementation in opntag and release it to a wider audience than the lab within a few weeks (on the hosted opntag.net site). Once we do so, we will conduct a survey and controlled tests comparing opntag's approach to relationship-based information sharing with that implemented in Facebook.

7. DISCUSSION

The representations of the data that emerged from our grounded theory analysis provide a set of propositions for understanding privacy requirements in social software. Our most important finding was that users have a fundamental assumption that when they put something in the tool, they should have control over it. Our data confirm the intuition that users can be reluctant to share personal information when the consequences of doing so are unclear, or when they are unable to control the transactional aspects of knowledge sharing activities. A counterintuitive consequence of this may be that some users are more ready to share personal information in a space that affords virtually no privacy control (e.g. blogs or Myspace pages) than one which offers them an inadequate set of privacy management tools. In our study, users were made aware that they could have some control of privacy and should manage the audience for their personal information by the promise of an access control system in the social system they used. When they found it inadequate, they often chose to not place information into the system because of the inflexibility of the tools or the lack of ability to model consequences of their actions.

This points to the importance of perceived affordances of privacy management mechanism for social software (as for any other user-oriented tool). As defined in the HCI field, perceived affordance is "action possibilities which are readily perceivable by an actor [11, 25]". Simply put, the concept emphasizes that suggested interactions with a tool must be in accordance with the ability of the actors to perceive those interactions. Perceived affordance has been identified as a major contributor to enhancing usability of a design [21, 24]. Because our privacy management mechanism is based on users' mental model of information privacy, we believe it provides better perceived affordance, thus improving the overall usability.

Even though the required privacy controls that were identified by our results were mere confirmation of the factors known by existing access control models, the fact that in social software these controls need to be in the hands of the users calls for new approaches in design of privacy management mechanisms in this context. We believe the insufficiencies of current mechanisms are the results of a significant gap between the perceived affordances of the underlying model and user requirements. We expect our findings to contribute to reducing that gap.

8. CONCLUSIONS

Although the use of social software for a variety of purposes has moved from leading edge to mainstream over the past few years, it is still in the early-adopter phase. Among issues that need further investigation are the issues of privacy and access management in these environments. We believe that the ability to understand and control information sharing in a natural, fluid manner is essential to the acceptance of these tools by a broad set of users, and yet, none of the existing access control models in the literature address the specific privacy needs of social software.

This research summarizes the results of our investigation into privacy issues as they pertain to the specific context of social software. We used the results of a grounded theory study of information sharing behavior to propose guidelines for the design of privacy control mechanisms. We discussed current access control models and explained why they are not sufficient for specific needs of social software, and then presented our proposed solution for a privacy management mechanism for social software that we believe can address those insufficiencies.

An important distinction between this study and previous investigations is how it goes beyond speculation to propose explanations as to why certain factors are important: our results are grounded in data gathered from users' experiences and opinions rather than deduced from the literature. As such, they give valuable insights into the processes entailed in information sharing in social software, and they provide a framework to direct further research.

It is yet to be determined whether our proposed solution has been successful in improving users' experience with the privacy

[1] In opntag, visibility implies readability, so there is no "I can see that it exists but can't read it" issue.

management mechanism. Clarifying where our solution stands compared to existing solutions (through performing usability studies) is part of our continuing research agenda.

9. REFERENCES

[1] Barkley, J., Beznosov, K., and Uppal, J. 1999. Supporting relationships in access control using role based access control. In *Proceedings of the Fourth ACM Workshop on Role-Based Access Control* (Fairfax, Virginia, United States, October 28 - 29, 1999).

[2] Boyd, D. 2006. Identity Production in a Networked Culture: Why Youth Heart MySpace., In *American Association for the Advancement of Science*, St. Louis, MO. February 19.

[3] Burrow, A. L. (2004). Negotiating access within Wiki: a system to construct and maintain a taxonomy of access rules, In *Proceedings of the fifteenth ACM conference on Hypertext and hypermedia*, Santa Cruz, CA, USA, pp 77 - 86.

[4] Cao, X. and Iverson, L. 2006. Intentional access management: making access control usable for end-users. In *Proceedings of the Second Symposium on Usable Privacy and Security* (Pittsburgh, Pennsylvania, July 12 - 14, 2006)

[5] Chakraborty, S. and Ray, I. 2006. TrustBAC: integrating trust relationships into the RBAC model for access control in open systems. In *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies* (Lake Tahoe, California, USA, June 07 - 09, 2006).

[6] Coulouris, G. and Dollimore, J. (1994), A security model for cooperative work, *Technical Report 674, Dept. of Computer Science*, Queen Mary and Westfield College, University of London, 1994.

[7] Erickson, T. From PIM to GIM: Personal Information Management in Group Contexts, *in Communications of the ACM*, January 2006.

[8] Fuller, S. (2002). *Knowledge management foundations.* Boston: Butterworth-Heinemann.

[9] Glaser, B., Strauss, A., *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Chicago, 1967

[10] Gross, R., Acquisti, A., and Heinz, H.J. III, Information revelation and privacy in online social networks, In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, p. 71– 80

[11] http://en.wikipedia.org/wiki/Affordance

[12] http://en.wikipedia.org/wiki/Web_2.0

[13] http://en.wikipedia.org/wiki/Principle_of_least_astonishment

[14] http://bluedot.us/

[15] http://del.icio.us

[16] www.facebook.com/

[17] http://www.flickr.com

[18] http://ma.gnolia.com/

[19] http://opntag.net

[20] http://www.orkut.com

[21] McGrenere, Joanna, Ho, Wayne (2000): Affordances: Clarifying and Evolving a Concept. In *Proceedings of Graphics Interface 2000*, May 15-17, 2000, Montreal, Quebec, Canada. p.179-186.

[22] Millen, D. R., Feinberg, J., and Kerr, B. 2006. Dogear: Social bookmarking in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada, April 22 - 27, 2006).

[23] Morse, J. M., Richards, L., *README FIRST for a User's Guide to Qualitative Methods*, Sage publications, 2002

[24] Norman, Donald A. (1988): *The Design of Everyday Things.* New York, Doubleday

[25] Norman, Donald A. (1999): *Affordances, Conventions, and Design*. In Interactions, 6 (3) p. 38-41

[26] Olson, J.S., Grudin, J., and Horvitz, E., A study of preferences for sharing and privacy, In *Proceedings of CHI 2005,* Portland, Oregon

[27] Patil, S, Lai, J. Who gets to know what, when: Configuring privacy permissions in an awareness application, In *Proceedings of CHI 2005,* Portland, Oregon

[28] Razavi, M. N. and Iverson, L. 2006. A grounded theory of information sharing behavior in a personal learning space. In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work* (Banff, Alberta, Canada, November 04 - 08, 2006)

[29] Resource Access Decision (RAD), *Object Management Group Healthcare Domain Task Force*, Revised submission, OMG TC Document corbamed/99-04-04, April 26, 1999.

[30] Sandhu, Ravi S., Coyne, Edward J., Feinstein, Hal L., & Youman, Charles E. (1996), Role-Based Access Control Models. Computer, Volume 29, Number 2, February 1996, 38-47.

[31] Shen, H. and Dewan, P. 1992. Access control for collaborative environments. In *Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work* (Toronto, Ontario, Canada, November 01 - 04, 1992).

[32] Sikkel, K. (1997a), A Group-Based Authorization Model for Cooperative Systems. European *Conference on Computer Supported Cooperative Work* (*ECSCW'97*), Lancaster, UK, 345–360.

[33] Sikkel, K. (1997b), A Group-Based Authorization Model for Computer-Supported Cooperative Work. *Arbeitspapiere der GMD 1055*, GMD, Sankt Augustin, Germany.

[34] Thomas, R., Sandhu, R., Task-based authorization controls (TBAC): Models for active and enterprise-oriented authorization management. In *Database Security XI: Status and Prospects*, North-Holland, 1997.

[35] Whalen, T., Gates, C., Private Lives: User attitudes towards personal information on the web, poster, *in SOUPS 2000*

[36] Whittaker, S., and Sidner, C. 1996. Email overload: exploring personal information management of email. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Common Ground* (Vancouver, British Columbia, Canada, April 13 - 18, 1996).

[37] Whitten, A., and Tygar, J.D., Safe staging for computer security. In *HCI and Security Systems Workshop*, CHI 2003, Ft. Lauderdale, Florida, April 2003.

Detecting, Analyzing and Responding to Security Incidents: A Qualitative Analysis

Rodrigo Werlinger,

University of British Columbia, Vancouver, Canada
rodrigow@ece.ubc.ca

ABSTRACT

Using grounded theory as qualitative method, this study provides a better understanding of the tasks performed by security practitioners during security incidents and the resources (tools, specific knowledge and skills) that these practitioners need to perform those tasks. The data for the analysis came from 24 questionnaires and 14 interviews of security practitioners mainly from academic organizations. The results show that a security incident can be separated in three stages linked by a temporal relationship: detection, analysis and response. Each stage is comprised by tasks that are performed using different skills, strategies and resources.

The paper also provides some recommendations about developing security tools. Central points of these recommendations are: Correlation of multiple sources of information, including the activities of different projects in distributed environments, and better trade-off between portability and visualization.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security Incident

Keywords

Security Tasks, Resources

1. INTRODUCTION

Security incidents are critical in the context of information security. To avoid them, organizations try proactively to apply preventive measures either to mitigate security risks, or to be prepared to respond to security incidents with minimal impact to the organization. Whatever strategy adopted, organizations need to invest a good part of their information technology (IT) budget solely on security: Two studies from Forrester Research, Inc. estimated this expenditure in over $100 billion USD in 2007 [14, 15]

Despite these efforts, security incidents are still frequent [10, 17, 18], implying negative and even disastrous financial consequences for organizations. Several studies and efforts have been devoted to report the cost of security breaches [19, 20, 21, 16]. To illustrate, a recent survey conducted in USA [21] revealed that, when a security incident implies the disclosure of customers' information, on average, an incident could cost to an organization USD$4.8 million.

Persistence and cost are the two factors that have motivated several studies about better practices for dealing with security incidents [24, 23]. Nevertheless, the literature is sparse concerning investigating IT professionals who have to deal with security incidents, in terms of which tasks they actually perform and which resources they need to face the complex scenarios given by real incidents [1]. This lack of research makes it difficult to evaluate and improve the support that IT security professionals need to not only handle security incidents, but also be efficient in their work.

This study aims at empirically investigating how security practitioners deal with security incidents by answering three central questions:

- *Which tasks are performed by security practitioners to manage security incidents?*
- *Which resources, in terms of skills, knowledge, and tools are needed by security practitioners to face security incidents?*
- *What tools used by security practitioners can be improved to deal with security incidents?*

In order to answer the previous questions this paper adopted an empirical focus, using ethnographic techniques [28] — questionnaires and interviews — to engage security practitioners' perspective during security incidents. These techniques enabled the study of security incidents in real contexts [7].

The ethical approval for contacting participants, the recruiting process, the interviews themselves and their transcriptions, were managed in the context of the HOT Admin project[1]. This project's field work provided 24 questionnaires and 14 interviews of IT security professionals with

[1]HOT Admin aims to devise a methodology for evaluating the effectiveness of IT security administrative tools, and to design effective technological solutions, guidelines, and techniques to aid security administrators [4].

responsibilities in IT security. The analysis started with the identification of the data with information about security incidents. To do so, a standard definition of security incident from the Computer Emergency Response Team (CERT) was used [3]. Afterwards, the chunks of text labeled with the theme *security incident* were analyzed using grounded theory [6, 5]. The open codes generated with this analysis were iteratively compared with the data. Axial and theoretical coding followed the open coding stage.

As a primary result of the analysis, the first two research questions of this study were answered by modeling: (1) the tasks performed by security practitioners during security incidents, and (2) the skills, knowledge and tools necessary to deal with security incidents.

The task model showed that the process of dealing with security incidents can be separated in three stages: detection, analysis and response. Detection accounted for all the tasks necessary for security practitioners to perceive anomalies in their environment. Then, the analysis stage comprised: (1) verification and assessment of the detection's outcome, (2) the tasks necessary to find out the source of the anomaly and track the source of the attack, and (3) interaction with other specialists. This last task was the most recurrent, supporting previous theories about the interdependency of security management tasks [11].

Three skills were used by the participants to perform their tasks: pattern recognition, generation of hypothesis, and collaboration. Knowledge about the IT infrastructure, protocols and attack patterns was used during security incidents. Tools used by the participants comprised a myriad of applications ranging from general IT to specific security tools to home made scripts.

Another result from the analysis was the discovery of another dimension of required resources to deal with security incidents: strategies. Strategies of isolation and simulation were mentioned as a way to find out the source of the incident and verify the existence of malicious software respectively.

The last research question was answered by providing recommendations about features that tools should have to support the work of security practitioners during security incidents. Correlation of multiple sources of information including the activities of different projects in distributed environments, and trade-off between portability and visualization are central points of these recommendations.

Another contribution of this paper is the comparison of the tasks modeled from the grounded theory analysis with those described in the standard RFC2350: "Expectations for Computer Security Incident Response" [22]. This comparison is used as reference for incorporating new questions in future interviews about, for example, the format of the reports used during security incidents.

The rest of the paper is organized as follows. The next section discusses related work. Section 3 describes the methodology, including data collection and analysis. Section 4 reports the results. Section 5 analyzes the results. Conclusions and future work are in Section 6.

2. RELATED WORK

Several lines of research can be distinguished from work related to security incidents. For the purpose of this study, these lines of research are separated in two parts. The first one, standard and best practices, summarizes standards and best practices developed by professionals working in the industry of IT Security. The second one, ethnographic and case studies about IT and IT security, covers IT and IT Security works that have used ethnographic methodologies to study IT administrators and IT security administrators.

Standards and Best Practices

Research and IT security communities have been prolific producing standards and best practices for dealing with security incidents [22, 24, 23, 8]. For example, the Computer Emergency Response Team (CERT), hosted at Carneggie Melon University, has published on its web site several resources about handling security Incidents [3]. Another example is the RFC2350: "Expectations for Computer Security Incident Response" [22], which sketches a framework about what to expect from Computer Security Incident Response Teams (CSIRTs). The same RFC defines CSIRT as the "team that performs, coordinates, and supports the response to security incidents that involve sites within a defined constituency." Despite the fact that these standards and best practices provide valuable information to organizations, in terms of designing and implementing policies and procedures to deal with incidents, none of them is intended to unfold the complexity of the tasks, interactions and processes that security practitioners have to face during a security incident.

Ethnographic studies about IT and IT Security

Barrett et al. [26] used ethnographic methods to study systems administrators. They used several quantitative and qualitative methods to gather information from IT administrators in large industrial service delivery centers. With 101 preliminary surveys, 12 interviews (sysadmins, managers, team leads, and others in various roles), 6 case studies (at 4 industrial service delivery centers), a log diary kept by a system administrator for 10 months (2002-2003), and observations of the tasks of 12 system administrators (over 25 days), they give several recommendations about developing tools to effectively support system administrators' tasks. Although Barrett et al.'s findings touch a broad spectrum of IT administration (e.g., database, web server, operating system), nothing is mentioned about specific practices, tasks and needs of administration in the domain of IT security. In addition, they are more focused on tool development, rather than providing models of the tasks performed by their participants.

Björck [2] empirically answered two research questions: *"What problems do organisations face and what processes do they go through as they are aiming to establish a balanced management system for information security?"* and *"What perceptions do information security managers hold as regards the management of information security in organisations?"* He used grounded theory to analyze the data that came from 29 semi-structured interviews: 8 with IT security managers, 13 consultants, and 8 auditors. All the participants belonged to Swedish companies. Björck's methodology is similar to the one used in this paper — an empirical perspective to answer his research questions. Nevertheless, his questions are more general: he endeavors to discover high level factors, models and perceptions that are involved in the design and implementation of information security management systems (ISMS); he is not focused on security incidents.

Kandogan and Haber [1] study aimed at evaluating security administration tools in real environments. They spent 40 days performing an ethnographic study of security administrators from a University in USA. Based on some real situations faced by these security administrators, they give recommendations about future developments of IT security tools. As Barrett et al., Kandogan and Haber do not provide models of the tasks and resources related to the participants. In contrast, my paper not only aims at providing recommendations about improvements on tools and resources used by security practitioners, but also obtains a deeper understanding of the task space and complexities during security incidents.

3. METHODOLOGY

The approach used in this study is based on ethnographic and qualitative techniques. The use of ethnography [28] makes it possible to study security practitioners in the context of security incidents within their organizations. To answer my research questions, this context is relevant because it not only accounts for complexities that could not be otherwise reproduced in a laboratory [7], but it also needs to be incorporated into the analysis for a better understanding of the phenomenon.

The ethnographic data were analyzed using grounded theory. Similar to Orlikowski [13], "the inductive, contextual, and processual" characteristics of grounded theory are used in this paper to construct an interpretation of security practitioner's view during security incidents. This constructivist posture also follows Charmaz's view about what means to build theories in grounded theory [5].

The ethical approval for contacting participants, the recruiting process, the interviews themselves and their transcriptions, were managed in the context of the HOT Admin project [4].

HOT Admin started with a field work study with 24 questionnaires and 14 interviews from IT professionals with different levels of IT security responsibilities. The profile of the participants and their organizations is described in the next section.

3.1 Data Sources

The 14 interviews from HOT Admin project comprised the main source of data for this study: thirteen of them mentioned IT incidents or IT security incidents. The interviews were accompanied by a questionnaire submitted by the participants, with general information about their responsibilities and technical background. The summary of this information is shown in table 1. All the interviewees came from British Columbia, and most of them (12) were from academic organizations.

Academic organizations have been the focus of other similar studies [1, 25]. The main reason for taking participants from academic organizations is they are easier to recruit than from other organizations. Undoubtedly, recruitment is a serious issue in IT security studies, as shown in [27].

One possible drawback of academic institutions is that the criticality of the information is not as high as in other fields (financial, commercial, military, etc). This could mean that sometimes IT security is not a priority, leaving some interesting IT security issues without further evaluation or investigation [25].

3.2 Data Analysis

The analysis of the data started with the identification of information about security incidents within the interviews .

A security incident was considered as : "any real or suspected adverse event in relation to the security of computer systems or computer networks " [3]. The aspects of security that were used to materialize this definition were confidentiality, integrity and availability [12].

About 13 situations related to security incidents were identified. This information was coded in an iterative process, starting with open coding and continuing with axial and theoretical coding [5]. At this stage, two models were generated. The first one comprised the tasks that the participants performed during security incidents. The second one accounted for the tasks in terms of resources (skills, knowledge and tools) necessary to deal with these incidents.

The posterior analysis was based on further elaboration of the "memos" [5] written during the coding process, which were initially used to draft any idea, comment or interesting finding from the data. Afterwards, these memos were used as the basis for the grounded theory results.

4. RESULTS

This section lists and classifies the security incidents mentioned by the practitioners. Further, the tasks performed by security practitioners are described. The section finishes with the resources used to support these tasks.

4.1 List of Security Incidents

Table 2 lists the incidents mentioned by the participants in different types. This classification takes into account the source of the incident, rather than its consequences.

The most common incidents reported were related to malicious software. Within this type of incident, our participants distinguished between specific types of malicious software (trojan, malware, worm), the quantity of compromised machines, the type of asset compromised (user's PC or internal Host), and the regularity of the event. The last three aspects were used as criteria to determine the way in which the incidents were handled. To illustrate, a large outbreak of virus required the participation of 20 specialists of different levels during more than 48 hours, whereas regular compromises of specific machines were covered by predefined procedures applied at the first level technical support service within the organization. As one of our participants said: *"If its infected [the machine] through ad-ware or that kind of scenario, then it just goes for re-image —that's our process...Usually I don't get involved if it's routine stuff."*

Incidents related to Human Resources were mentioned in terms of the violation of internal policies. These violations were related with improper use of the organization's resources. These incidents were also characterized for the sensitivity of the internal communications during their investigations: *"if it's dealing with a sensitive nature where people's reputation could be damaged, that kind of stuff, then it tends to come directly, and it's quietly, whereas the rest of them tend to come through the help desk."* Incidents related to launch attacks against third parties make it necessary to be aware not only of the incoming traffic, but also of the outgoing.

Phishing was a type of incident mentioned by one of the two participants in the private sector. This was handled by

Table 1: Demographics of the interviewed participants. N/A accounts for not available. This happened because participants were not obliged to answer all the questions from the questionnaire.

Job Position	Organization type	% of time spent on IT security	Security training
Security Specialist / Business Continuity Process Specialist	Banking	N/A	N/A
Tech Specialist II	Insurance	20	yes
Network Security Manager/System Administrator/Videoconferencing	Academia	10	yes
Application Programmer		40	N/A
Director, IT Services		N/A	No
Information Security Officer		N/A	yes
IT Security Officer		N/A	yes
Senior Systems Analyst		25	N/A
Senior Systems Analyst		N/A	yes
N/A		N/A	N/A
Security Analyst		N/A	yes
Systems Administrator		60	N/A
Network/Security Lead		40-60	yes
Systems Analyst		20	N/A

only one person. It had different characteristics from the other incidents, and was classified in a different category.

Suspected security incidents includes those incidents that either were being investigated and there was no clarity about their causes, or those incidents that could materialize serious compromises in the future. In the former case, two of the participants reported situations where the source of the problem was not clear, and they had to speculate about the presence of a malicious source. One of them specified: *"So we try to put a proxy in between —a very powerful Linux machine— and then it started crashing. So like I said: I don't think it's malicious, it's all firewalled away."* These incidents were interesting because the participants needed to perform more tasks and use more resources and skills to discover the source of the problem. In the latter case, port scanning was an alerting signal for the participants, in terms of expecting a security incident if they did not address the situation. One participant commented about port scanning: *"They're basically just probing and looking for an open port, so you say yep, this person's probing. You just take their IP address, you send a complaint to their network administrator..."*

Table 2: List of security incidents

Description	Incidents [number of occurrences in the data]
Malicious SW	1. Host infected with a worm [1] 2. A user's PC with Malicious Software [3] 3. Large outbreak of virus [2] 4. A Host with a Trojan [1]
Human Resources	5. Download porn [1] 6. Hack other systems using organization's infrastructure [1] 7. Send threats emails from organization's servers [1]
Phishing	8. One case of phishing reported by a client [1]
Suspected incidents	9. Peaks of traffic [1] 10. Unreachable systems [1] 11. Devices crashing [1] 12. Network slow [1] 13. Port scanning [1]

4.2 Tasks

Table 3 shows the main tasks performed by our participants during the security incidents. These tasks were grouped in three main stages: Detection, Analysis and Response. These stages account for the temporal sequence since a security incident is "perceived" by the security practitioner until a concrete action to stop it is taken. In between, during the analysis, security practitioners have to perform several tasks to confirm the incident, assess its scope, and find out the source of the problem or the attack. In section B is shown in detail the temporal relationship of these tasks for the incidents described by the participants.

Monitor systems and networks: The objective of this kind of task was to detect incidents by either direct inspection of systems and networks, or by using SW tools that detect anomalies in the systems' behavior. This kind of task was common for all types of incidents.

Receive notifications: This kind of task was also common for all types of incidents. Some notifications came from third parties external to the organizations, as the participant who dealt with the phishing attack explained *"...we had a person, not even a member of any of our organizations or customers, who emailed our privacy office."* Another example comes from a participant who described how his organization received notification e-mails from myNetWatchman, an external IDS system, that detected anomalies in remote networks: *"This morning it happened that I got an e-mail through myNetWatchman, which was relating to basically a worm on a machine here."*

Other notifications came from users who reported their problems to the helpdesk. From here, these complains were communicated by automatic reporting systems to the participants, like tickets or e-mails: *"as a general rule a lot of the calls for stuff like viruses and problems with the computer all go to the help desk...we have an automated incident*

Table 3: List of tasks performed during a security incident

Stage	Task
Detection	Monitor systems or networks Receive notifications
Analysis	Assess the incident Verification Track the source of the attack Find out source of the problem Coordinate with other specialists Generate action plan Evaluate legal implications
Response	Turn off ports or services Clean-up systems Re-initialize services Patch or reconfigure systems System's restoration Administrative sanctions

monitor system that is our helpdesk automated system."

Assess the incident: Assess the incident is the first task in the line after detecting the incident. The primary objective of this kind of task was to evaluate the incident, in terms of its magnitude and possible consequences.

One possible output from assessing the incident was to re-prioritize tasks and start handling the incident before other pre-scheduled tasks: *"If it looks like a compromise, I might go through the logs to see what kind of traffic I'm getting from this IP address — everywhere else in the campus; is it scans; is it a successful compromise. So it depends on what I find, depends on what I do."*

Other possible outputs after assessing the incident were either to assign or reassign a coordinator for dealing with the incident or generate internal notifications. Usually it was a helpdesk in charge of redirecting incident notifications: *"they [the help desk] would reassign the open-call... we have an automated incident monitor system that is our helpdesk automated system, so they would route it back to his queue as an open call."*

Verification: The main objective of this kind of task was to confirm that the information from the detection stage (either from notification or monitoring) was accurate and there was effectively a compromise (i.e. not a false positive). To do this, participants either checked directly with the people responsible for the suspected machine, or used log files from other systems and tools to perform their own analysis. One example of the first comes from a participant from academia. This person did not manage all the information of the projects on the IT infrastructure: *"If a machine has way more traffic than a web server or some other kind of high-use server, then there's something funny going on with that machine, and then we trace it back, and sometimes it's [someone's legitimate]project..."* The same participant described the use of different sources of data for verification purposes: *"I always try and verify by a second or third source. So [I would, again] go back to the Argus...check the Argus logs and see what's actually happened; what have I seen in terms of the traffic going towards that machine; what ports is it going after; what's the size of the data. Then I would actually go to one of my other logs [say from a] windows box; what have I seen in the logs of the windows box; was that a real compromise or not."*

Another case of verification also required interaction with other specialists and the execution of specific subtasks such as finding the responsible of the suspected machine, and tests in-situ: *"Resolved the IP address to a name and looked up the name in our site data base to see who is supposed to be responsible for the machine, and then forwarded an e-mail to that person and [to] the person that looks after that Windows machines on site."*

Track the source of the attack: The participants rapidly recognized some incidents as attacks. In this case, just after the notification, the participants started tracing the originator of the malicious activity. For example, in the case of the phishing, the participant used the information from the e-mail that notified about the phishing attack to find where the web page that impersonated the web page of his organization was hosted. He was able to trace back the source of the attack to a host in Germany. At this stage, he had to interact with the administrator of the service provider that owned the IP of the server, to block access to the fake web page.

Find out source of the problem: Some incidents required more analysis from the participants to find the source of the problem. Usually this was a case of detection by monitoring, where the monitored systems showed symptoms of being compromised without further information. Some incidents triggered by malicious SW were characterized by this kind of task. For example, one participant mentioned how difficult was to trace an infected machine source of a DoS attack: *"One machine can create a Denial of Service for everyone on the network —and then to [find its source is difficult] because [the network is] so overloaded."*

Coordinate with other specialists: The coordination with other specialists was necessary either to complete information about the incident, or to come up with specific plans of action or to investigate.

Evaluate legal implications: This kid of task was unique for incidents related to policy violations. When an organization's resources were used as a platform to attack other organizations, it was necessary to interact with legal experts to analyze the legal responsibilities of each party involved.

Generate action plan: An action plan was the main output by a group that analyzed the large outbreaks of viruses. This action plan was the result of the analysis of the virus, and consisted of a procedure to stop the the virus and clean the affected machines.

Turn off ports or services: This was a usual action taken for isolating the source of an incident.

Clean up systems: Usually for incidents due to malicious SW, the infected machines were just cleaned up.

Re-initialize services: This kind of task was performed when configurations of certain devices were suspected to be corrupt. This kind of task was performed when the cause of the incident was not clear.

Reconfigure systems: This kind of task was performed particularly when a firewall had been configured to send its log files to a syslog server. As this traffic made the network collapse, the firewall had to be reconfigured.

System restoration: This kind of task takes a system backup to restore the service. In some cases the task was very complex, because it was necessary to perform several tests to check that everything was working. As one participant explained: *"I came in — we had had a serious VPN*

server failure on Friday, so I had actually to bring on a secondary unit, and there were some serious problems associated with bringing that unit on-line..."

Administrative sanction: This task was unique for cases of internal policy violation, and was related to the sanctions of the persons involved in the incident.

4.3 Resources

To perform tasks during security incidents, our participants needed to make use of several resources that included SW tools, specific knowledge, skills and strategies, as described below.

4.3.1 Tools

SW tools comprised a resource. A recurrent example was the use of Shell/Perl scripts written by themselves to monitor systems. These scripts looked for specific patterns of suspicious activity in firewalls and IDSs' log files. They then generated automatic e-mails to those responsible for handling the incidents. As one participant said: *"I have scripts that go off of the logs to analyse stuff there. So the first thing is to check the e-mail and, if there's any alert in them, it could be that some machines are issuing to much; there's probably virus activity. We have flags for that."*

There were also specialized tools to monitor virus activity. Two participants said they used McAfee EPO to get reports of quantity of viruses per machine: *"We've got MacAfee. That's another one, partly, that we use with the EPO. It [reports] all the viruses too, so I can keep track at machine level."*

To receive notifications, participants needed either an e-mail client or a connection to the system used by the organization to generate tickets via the Helpdesk. In the case of external parties sending notifications, there was necessary an inventory of the parameters of network's configurations (autonomous systems (AS) or IP addresses space) with the contact information of those in the organization who were responsible to configure such parameters.

To analyze the packets of the network and find out the source of the attack or the problem, tools like TCPDump and Ethereral were used. In this case, a participant had to not only know how to use these tools, but also to have knowledge about filtering techniques to reduce and extract those parts of the files that were useful for the investigation: *"I mix and max between TCP Dump and ethereal. So TCP Dump can read the binary file and a can re-write another binary file after applying a filter."*

To clean-up systems, antivirus and spywares were used. Automatic updates were often used to patch systems.

4.3.2 Specific knowledge

To verify a suspected compromise, participants needed to compare other sources of information. Sometimes, this information came from other people, who were contacted to check what was happening. Besides a phone and an e-mail client, participants needed to know the contact information of other specialists in charge of IT infrastructure. This inventory was critical in academic organizations, given the distribution of responsibilities around the IT infrastructure.

In the case of the phishing incident, the participant used both his experience in configuring networks, and specific knowledge about phishing attacks. He actually had a plan prepared by himself beforehand to respond to phishing incidents.

To generate hypothesis, the participants needed to know the IT infrastructure, the ways that the systems interacted and specific knowledge about protocols. To validate the hypothesis, they needed to know how to analyze the traffic that traversed the network. As this traffic is not stored, one participant needed to write a script to take advantage of the network capabilities and mirror the packets of the affected area to mount another server to capture and store those logs: *"I was creating a script that basically would capture the packets that are coming, the mirrored packets, and placing it in a file, putting a time stamp, and archiving it after two days or three days when we no longer need it."*

4.3.3 Skills

Pattern recognition: Pattern recognition was a recurrent skill that our participants had to use specially during the first tasks to handle the incidents. One participant used this skill during the detection stage, setting a predefined threshold in the number of e-mails per machine. He knew that more than a certain number of e-mails per machine corresponded to virus activity patterns: *"machines that are kicking out more than 50 e-mail connections an hour [gives]us an indication of virus activity..."*

Other participants described how they were able to match DoS attacks with predefined patterns. One participant said *"Denial of services are easy to spot, cause its sending millions of the same thing, actually over and over and over again, with very little iteration or very little permutation."* The other stated: *"This is a typical pattern of a hacker. They get into a machine through some exploit or some way they have in, and then they start downloading stuff with WGet or Curlo or things like that..."*

The participant in charge of handling the phishing attack described how he tried to find a typical pattern when he was assessing the magnitude of the incident: *"Usually when there's a large phishing event, there are a lot of emails sent out —your email server is innundated with non-deliverable, because its like spam..."*

Hypothesis generation: Two participants had to generate hypothesis about those incidents where the cause of the problem was not clear. One of them described: *"What we are trying to do is that we have noticed in the past that we've gotten spikes in traffic to one of our production servers and we haven't been able to trace what the spike is due to...We think that there has been a breakdown in TCP/IP connections between our router and our server."* About the same type of incident, one participant hypothesized: *"So we suspected the network and then we both sort of came to the conclusion it had to be upstream."* During the investigation of a proxy device crashing without apparent reason, a participant said: *"As soon as we put the bridge in, it stopped crashing without filtering. It just should be a transparent network device for all valid packets, so we hypothesize that it was a malformed packet."*

Another participant generated hypothesis around the way that a machine was infected with a malicious software: *"That particular machine has a running VNC server so one possibility is that somebody is able to scan that and guess a password for the VNC server. Or another possibility is that somebody just downloaded a Trojan on the machine."*

Collaboration: Collaboration with other specialists was recurrently mentioned by the participants. The motivations

for this cooperation were diverse: make a more efficient investigation, execute specific actions in-situ, gather network information and design a response plan to clean-up systems infected by a virus.

To investigate what was causing an incident, a participant interacted with a colleague *"because two eyes are better than one..."* Another participant remarked about the positive results after interacting with other specialist: *"Between the two of them we finally isolated — hey, its that new firewall that we just brought up."*

In distributed environments it was necessary to get from other areas information about the physical location of compromised machines: *"So I would pass it over to the network guys and they'll find out where the IP address sits."* The same participant explained how other specialists were required to clean-up systems or disconnect ports in-situ *"They [network guys]'ll forward it off to our helpdesk people to clean it up, and so they'll actually go out and clean it up "* and *"they actually logically unplug it so they disable the port."*

During the phishing attack one participant had to interact with an administrator in Germany to take down the false web site. Another participant explained a more complex interaction during a big incident because of a virus. In this case, there was a group of 20 people that had organized in two different subgroups: *"There were a few of us who just kind of kept up with what was going on, tracked all the decisions that were made, the status of where we were, how many services were being disrupted, how many systems were infected, kind of kept track of where we were, and while the rest of them at the time had to go out desktop to desktop."*

4.3.4 Strategies

The concept of strategies for dealing with security incidents emerged unexpectedly through grounded theory analysis. Below two examples are described.

Isolation Isolation was a strategy used to either verify incidents or to find out what was causing the anomaly or the attack. Examples of the former case were already mentioned in the description of the verification task. For the latter case, one participant who was investigating why the internet connection was slow stated: *"We also contacted IT services [to] see if they could see, based on traffic utilization on the network, where it was coming from...we finally isolated — hey, its that new firewall that we just brought up."* This example shows how, by comparing observations with other specialists, the source of the problem was isolated.

Simulation To investigate security incidents, participants sometimes needed to simulate the compromise, either in a controlled environment or in the production network. In the interviews, these simulations had the objective to either verify the existence of malicious software in a compromise, or get more evidence and clues about the the source of the incident.

To illustrate the first case, one participant explained how he downloaded the same suspected malicious software to check it: *"It's saying...downloading a tool from some website. Okay, so I do that, download this tool and run it through the antivirus and it says okay, this is some dialup ..."* In the second case, another participant mentioned how he was trying to collect information from real situations where he repeated the conditions of failure: *" So we try to put a proxy in between...and then it started crashing...as soon as we put in no filtering...bad things stop happening..."*

5. DISCUSSION

The previous section described the main results from the grounded theory analysis of the interviews. This section discusses those results in terms of: (1) what to expect from security incidents; (2) a comparison of the task model from this study with that from a specific standard; (3) relationship between the results of this study and the concept of interdependency of IT security tasks; and (4) recommendations for improving the tools used by security practitioners focused on their security tools.

5.1 What to expect from a security incident

Our participants described 10 security indents in detail, and 3 partially. Although this fact is highly determined by the types of questions during the interviews, all the participants mentioned at least once to be involved in security incidents. This shows that handling security incidents is a common activity of security practitioners.

The descriptions given by the participants ranged from specific characteristics that made their environments complex in terms of IT security (see appendix A), to the consequences that a security incident could have within their respective organizations (see appendix C). Although investigating what makes security practitioner's work more difficult or easier was out of the scope of this study, the two characteristics of decentralization and academic freedom may make the results of this study extensible to other organizations which are not regulated in terms of security. The consequences of security incidents described by the participants seem general enough to be extended to other organizations, although more validation is required.

Another important result from the analysis is that a security incident can be separated into three phases: detection, analysis and response. Each one has its own tasks and resources, and accounts for the temporal sequence of events since the incident or suspected incident is *perceived* until a concrete action is taken.

The results showed no strong correlation between the security training specified in the questionnaire and the specific tasks performed or resources used by the security practitioners. Only in the case of the phishing attack, was training explicitly highlighted as an asset to handle that type of incident.

The results also showed that not all the incidents required that the participants perform the same tasks or use the same skills, strategies or resources. This information is useful to "predict" what to expect from different types of security incidents. On the one hand, we had incidents where it was clear what to do after the notification. For example, the phishing incident had a very defined pattern that was quickly recognized to start finding the source of the attack. The same happened with incidents related to violations of internal policies, and to malicious software in user's PC.

On the other hand, other incidents are expected to have a stronger analysis component. This happens when the information from the detection stage is not specific, or only accounts for general symptoms of the problem. The response to these incidents would be quicker if there were tools able to detect more specific patterns of attacks and better reporting capabilities.

Another aspect to consider in handling security incidents is the way that people interact and the different roles involved. There was no incident in which the same person

performed all the tasks from detection to response. Usually the information from the detection stage was concentrated in the person who assessed or verified the incident. This person could be someone from the helpdesk or the same security practitioner. In the case of the helpdesk, other notifications were generated internally to the predefined coordinators or *"owners"* of the incident. The criteria used for assessing the incident in this case were not explained during the interviews, and is one of inputs for the next step in this study (see D section).

After the assessment or verification there were other interactions already explained in section 4.3.3. Some of these interactions were with specialists who actually performed the final tasks in the response chain. Some of these specialists would need to be interviewed in order to unfold the complexity of response tasks.

5.2 Contrast with RFC2350 standard

Standards and best practices are intended to provide general recommendations about handling security incidents. They provide an overview of the steps that a CSIRT should follow to content and eradicate an incident.

One representative example of these standards is the RFC2350: "Expectations for Computer Security Incident Response", whose purpose is *"to express the general Internet community's expectations of Computer Security Incident Response Teams (CSIRTs)."* As a standard, the RFC2350 does not intend to propose a model that comprises tasks, skills, strategies and resources used during an incident as the present study does. Nevertheless, it is possible to draw a comparison based only on the tasks resulting from the qualitative analysis and on the list of services the RFC2350 proposes a CSIRT should provide to respond to security incidents.

Both the RFC2350 and the results of the present study recognize the presence of tasks related to assessment (report assessment in the RFC), verification, interaction with other specialists (coordination in the RFC) and restore the system (recovery in the RFC). Although forced, a match may be established between the tasks "Track the source of the attack" and "Find out the source of the problem" from this study and "Technical assistance" from the RFC. The same exercise may be done between the tasks "Clean-up systems" and "Reconfiguration of systems" from this study and "Eradication" from the RFC.

The tasks related to the detection stage described by our participants are not mentioned in the RFC. In the other direction, "Information categorization" (categorization of the incident related information with respect to the information disclosure policy) was not a task identified in the results of my analysis. Future interviews should provide more information about this difference.

Another aspect to explore in future interviews is the use of formal reporting templates for registering security incidents; the participants did not make reference to the use of specific reporting procedures during the interviews. Future interviews will be used to obtain more information about this difference.

5.3 Interdependence

Knapp et al [11] reported high interdependence of information technology security tasks from a survey performed on 936 certified information systems security professionals (CISSPs). This interdependency refers to "the extent to which individuals depend upon other individuals and resources to perform a job." My results confirm the Knapp et al findings in the context of handling security incidents. This conclusion comes from noticing that the most common task performed by the participants during the analysis stage was the interaction with other specialists.

5.4 Better security tools

Within the resources used to handle security incidents, SW tools played an important role. Some of them were specific security tools, whereas other belonged to the IT world in general. Examples of the first ones were: Kasperski and McAfee antivirus, IDSs as Snort and Argus, and Firewalls' administration software. In the group of the second ones participants mentioned TCPDump and Ethereal.

SW tools were usually complemented with scripts written by the same participants. Sometimes scripts aimed at not only complementing but also replacing the functionality of well known tools. For example, one participant used only his scripts to monitor the systems, discarding the information from the IDS snort because of the big quantity of false positives. The same participant expressed his need about a specific tool able to monitor suspected SQL queries on the databases.

Another participant used TCPDump and ethereal sequentially to generate and analyze the log files he needed for his analysis. He alternated between the advantages of portability (TCPDump) and good visualization (Ethereal): *"[TCPDump provides] common analysis format... it's also a portable format... [Ethereal runs] colours things...it shows the SYN and Reset in one colour and then the Push commands in another colour — so it is obvious — there is content in there."*

The experiences described by our participants could be improved by tools that:

- Monitor the networks and systems correlating other sources of information as project's inventories to discard false positives. In this case the tool should correlate not only with the protocols that are in the network, but also with the list of projects that affect different IT infrastructures in the distributed environment. The tool in this case should help the practitioner to discriminate what activity comes from a project from what's not.
- Monitor specific systems and specific protocols, as SQL queries.
- Integrate scripts on them.
- Are able to use as inputs different types of files and generate portable outputs, with good visualization. This tool would be used to analyze log files from different sources.

6. CONCLUSIONS AND FUTURE WORK

This study showed the results of a qualitative analysis on how security practitioners deal with security incidents. The data for the analysis came from 14 interviews of security practitioners, mostly from academia.

The results of the analysis comprised a list of incidents, a model for the tasks, the skills and the strategies used during security incidents.

The discussion of these results contrasted the results of the analysis with related work and highlighted from the analysis those aspects that enable prediction of what to expect from security incidents.

The qualitative analysis performed represents the first step in applying grounded theory. The next step in this study is to gather more data to refine the results shown in this paper. To do so, more security practitioners need to be interviewed about incidents. These practitioners should come from fields different from academia.

7. REFERENCES

[1] Eser Kandogan and Eben M. Haber, "Security Administration Tools and Practices", Security Administration tools and practices, O'Reilly, August 2005, chapter 18, ISBN: 0-596-00827-9.

[2] Fredrik J. Bjorck, "Discovering Information Security Management", Doctoral Thesis, Stockholm University / Royal Institute of Technology, Report series No. 05-010, ISSN 1101-8526

[3] Computer Emergency Response Team (CERT): Computer Security Incident Response Team (CSIRT) main page (2007), available at: http://www.cert.org/csirts (accesed February 2007)

[4] HOT Admin project (2007), available at: http://www.hotadmin.org (accesed February 2007)

[5] Kathy Charmaz, "Constructing Grounded Theory", SAGE publications, 2006.

[6] Barney Glaser and Anselm L. Strauss, "The Discovery of Grounded Theory, Strategies for Qualitative Research", Aldine Publishing Company, Chicago, Illinois, 1967.

[7] Qualitative Research in Information Systems (2007), available at: http://www.qual.auckland.ac.nz/ (accesed February 2007)

[8] Computer Emergency Response Team (CERT): Main Page (2007), available at: http://www.cert.org/ (accesed February 2007)

[9] Bagchi, K. and G. Udo, "An Analysis of the Growth of Computer and Internet Security Breaches", Communications of the AIS, 2003. 12(46): p. 684-700.

[10] Gordon, Lawrence A., Loeb, Martin P., Lucyshyn William and Robert Richardson, CSI/FBI Computer Crime and Security Survey, 2006, available at: http://www.gocsi.com/ (accesed February 2007)

[11] Kenneth J. Knapp, Thomas E. Marshall, R. Nelly Rainer and F. Nelson Ford, "Managerial Dimensions in Information Security: A Theoretical Model of Organizational Effectiveness", research report prepared for the $(ISC)^2$ Constituency, October 25, 2005, available at: https://www.isc2.org/download/auburn_study2005.pdf (accesed April 2007)

[12] Wikipedia, CIA triad web page definition (2007), available at: http://en.wikipedia.org/wiki/CIA_triad (accesed April 2007)

[13] Orlikowski, Wanda J., "CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development", Management Information Systems Quarterly, Vol 17, No. 3, September, 1993

[14] Khalid Kark, Jonathan Penn, Christine E. Atwood, and Jennifer Albornoz Mulligan "The State Of Information Security Spending, Information Security Spending Trends Downward", Forrester Research, Inc., 2006

[15] Bartels, A. and Holmes, B. J. and Lo, H., "Global IT Spending and Investment Forecast, 2006 to 2007", Forrester Research, 2006.

[16] Garg, A., J. Curtis, and H. Halper (2004), "Quantifying the financial impact of IT security breaches?" Information Systems Security, 2004. Vol. 11 No. 2, pp. 74-83.

[17] Web application security consortium, main page (2007), available at: http://www.webappsec.org/projects/whid/statistics.shtml (accesed April 2007)

[18] National ICT Security & Emergency Response Centre (2007), available at: http://www.niser.org.my/statistics/2006.html (accesed April 2007)

[19] The Fifth Workshop on the Economics of Information Security (WEIS 2006): Main Page (2007), available at: http://weis2006.econinfosec.org/ (accesed February 2007)

[20] "National Survey on Data Security Breach Notification", PGP Research Report Summary, November 2005, available at: (accesed February 2007)

[21] "2006 Annual Study: Cost of a Data Breach Understanding Financial Impact, Customer Turnover, and Preventative Solutions", PGP Research Study Summary, October 2006, available at: http://www.computerworld.com/pdfs/PGP_Annual_Study_PDF February 2007)

[22] Brownlee, N., Guttman, E., "RFC 2350 - Expectations for Computer Security Incident Response", June 1998, available at: http://www.ietf.org/rfc/rfc2350.txt (accesed February 2007)

[23] TCP/IP Network administration: chapter 12. Network Security (2007), available at: http://www.unix.org.ua/orelly/networking/tcpip/ch12_01.htm (accesed February 2007)

[24] Forum for Incident Response and Security Teams: Main Page (2007), available at: http://www.first.org/ (accesed February 2007)

[25] Sherri Davido and Bob Mahoney, "Incident Response and Large Event Handling in the Research University", 16th Annual FIRST Conference on Computer Security Incident Handling, Budapest, Hungary, June 2004.

[26] Barrett, R. and Haber, E. and Kandogan, E. and Maglio, P. and Prabaker, M. and Takayama, L., "Field studies of computer system administrators: Analysis of system management tools and practices", Proceedings of the Conference on Computer Supported Collaborative Work, 2004.

[27] Andrew G. Kotulica, Jan Guynes Clark, "Why there aren't more information security research studies", Information & Management 41 (2004) 597607.

[28] Fetterman, David M., "Ethnography, Step by Step", Applied Social Research Methods Series, Volume 17, second edition, 1998.

Table 4: Flow of tasks for some security incidents

Incident	Sequence of tasks
1	D1 - A1 - A2 - A5 - R2
2	D2 - A1 - R2 or D1 - A1 - A5 - R2
3	D2/D1 - A1 - A4 - A5 - A6 - R2
4	D1 - A2 - A4 - R2
5	D2 - A1 - A7 - R6 or D1 - R6
8.	D2 - A1 - A3 - A5 - R1
9.	D1 - A4
10.	D1 - A2 - A4 - R5
11.	D1 - A2 - R5
12.	D2 - A1 - A4 - R4

APPENDIX

A. THE WORKPLACE

Most of the participants came from academic organizations. In terms of IT security, participants described this environment as particularly challenging for two reasons: decentralization of responsibilities and academic freedom.

Regarding decentralization of responsibilities, a participant recognized that decentralization has its advantages, but they are at the expense of security: *"the decentralized nature does not help. Decentralization can bring particular strengths and can bring flexibility and agility in certain things, but those are often at the expense of security."*

About challenges related to academic freedom, one participant stated: *"Some of the things that would be best practices in the security suite of tools we can't necessarily implement quite as easily as a private sector organization can, simply due to the academic freedoms and expectations and needs of faculty and students. I know that's an interesting trade off all the time. You're constantly trading access versus risk."* Nevertheless, academic freedom was also considered a positive attribute in terms of trying new solutions for IT issues: *"I think basically because of the fact that we have that academic freedom...I think I could be a little more creative here than I could in another organization, so, for example, I could have a server of my own set up exactly how I want set up with all my own tools on it ... without having to make a case for it, or without having to say everybody is using this shell and this type of box, I can do whatever I want to do."*

The two interviews from participants that were not from academic organizations mentioned challenges related to trading-off security and the business model, and getting privileged information from internal databases. Nevertheless, this is preliminary, and it will be developed in the next steps of the study.

B. TEMPORAL SEQUENCE OF TASKS

Figure 1 shows the relationship in time of the tasks performed during security incidents. Table 4 shows in detail the sequence of tasks for the incidents described by the participants.

C. CONSEQUENCES

One possible consequence of a security incident is the revision of policies: *"Other times it's when security incidents would arise and we would look to see, is there a policy to cover this? or we would need to interpret our current policy in light of an incident, and we find that there either wasn't a policy that covered that particular case where there needed to be."*

Another consequence is that security issues become a first priority: *"They're most responsive after an incident, not before. So we generally have a two week period after something major happens, where you've got a two week window to get some stuff pushed forward as top priority, and then it's bottom priority."*

D. TOPICS TO REFINE THE RESULTS

1. Sequence of events followed during a security incident. Possible question: Could you describe the sequence of events during the security incidents you have been involved?, How it was detected, analyzed and solved?
2. Roles of people involved in the incident. Possible question: Could you detail who was involved in the incidents and which role they played?
3. Decisions they had to make during the incident. Possible question: Which decisions had to be made during security the security incident?, which criteria were used? there was any standard?
4. Resources. Possible question: Which tools or infrastructure was necessary to handle the incident?
5. Validation of the information. Possible question: Do you have a written procedure to handle security incidents?, Would you be willing to be observed when handling a security incident?

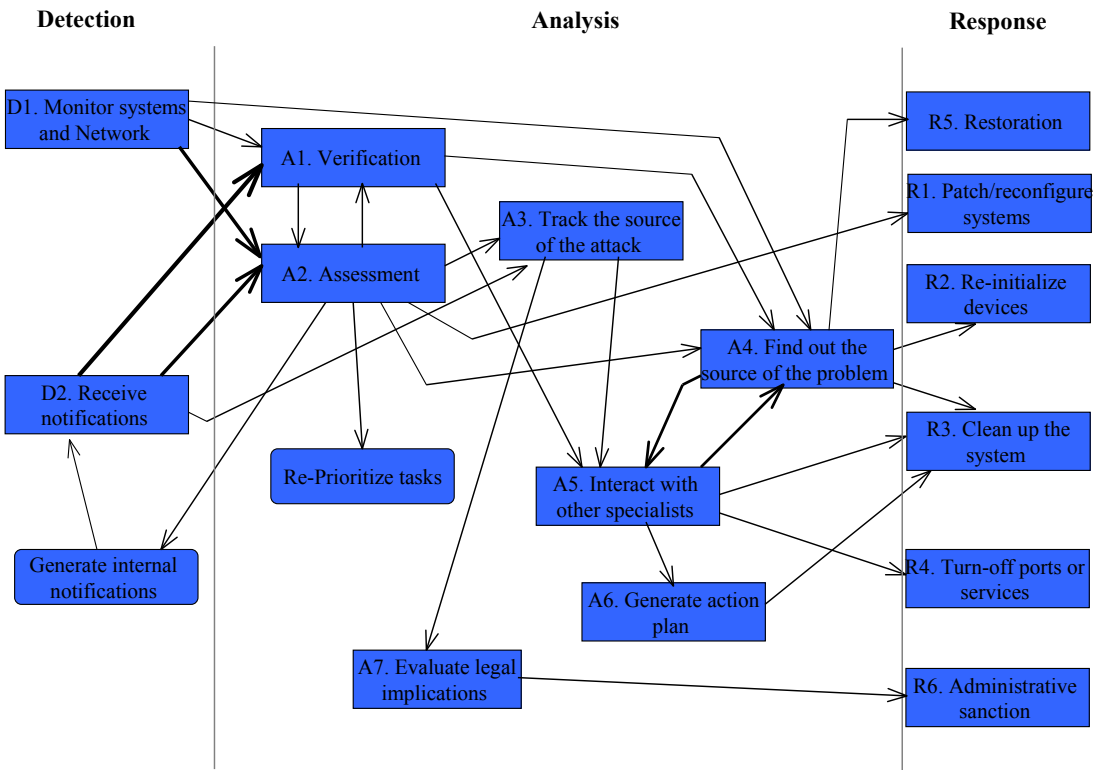

Figure 1: *Sequence of tasks during security incidents. Thicker arrows represent more common transitions*