Area 2 (Programming Techniques and the Web)

Danny Yoo (dyoo@cs.wpi.edu)

Abstract

The object capability (ocap) community uses Y-URLs (yurls) as a means of extending the ocap paradigm to the distributed world of the Web. Loosely, a yurl is a unspoofable name for a resource. This is reminiscent of the URLs generated by the continuation-based PLT Web Server, but is it the same?

1. Compare and contrast yurls and the URLs generated by the PLT Web server. In what ways do the two differ, and what impact do these differences have on ocap properties?
2. The PLT Web Server also offers a "stateless" mode. How do the URLs in this mode compare against yurls, and again, what impact does this have on ocap properties?

1 Introduction

A URL is a reference to a resource; for web servers that produce content on-the-fly, a URL does not only point to static data, but also acts as a name to a pending computation. This connection between URLs and computations makes a URL a potentially viable representation for an object capability, with the Web as the fabric for a distributed object-capability security system.

However, the textual transparency of URLs poses an immediate forgeability problem that threaten to break the properties of the object capability model. This article investigates the use of computation-associating URLs in two web server frameworks: the Waterken server [3] and the PLT Web Server [7], and compares how their different URL encodings affect their applicability as object capabilities.

2 Object capabilities and URLs

An object capability system is an application of the actor model [6] toward security policies. In the regular actor model, actors act by sending messages to other actors in the system. In the absence of global state, an actor can only use its local knowledge to send messages to those it knows about. A message can hold references to other actors as an act of introduction.

An object capability-based security model adds a security semantics to this model by coupling the act of object message passing with that of exercising a permission. An object can perform a privileged operation by sending a message to the appropriate target object. Outside of message passing, privileged operations are, by default, denied. The act of introduction becomes one that delegates permission. The permissions granted to an entity, then, can be considered as the set of messages that it can send.

2.1 Object capability properties

The object capability model hinges on the following properties:

- unforgability: object capabilities can not be manufactured without prior knowledge.
- transferability: although they can't be manufactured from scratch, objects can be copied and delegated off to another party.

- subdivisibility: if an object performs a action with rich behavior, another object can be used to wrap around them to limit that behavior. e.g. if an object represents file system access to any subdirectory, a wrapper can interpose a filter to limit access to a fixed set of subdirectories.
- revokability: the originating entity should have the ability to revoke a permission it delegates to another entity, since it has a reference to that object.

In the context of distributed computing, a distributed object-capability system adds a security requirement to messages passing:

- secrecy: the mechanism for message passing from sender to receiver must ensure that the content of the message is unobservable to others. Otherwise, an external party will be able to observe the communication and gain a permission that it has no right to have.

2.2 URLs in Waterken and PLT Web Server

In the context of dynamic web servers, a URL acts as a trigger to activate a pending computation on the server side. In the language of capabilities, a URL is a message whose source is the client browser and whose target is the server-side computation. It's tempting to adopt a distributed object capability system by using URLs as the messaging mechanism. URLs are immediately transferable due to their textual transparency: they're embeddable in email and web pages, and can be copied by inspection. On the other hand, the other ocap properties that we'd like to hold—unforgability, subdivisibility, revokability, and secrecy—can't be considered without looking at them in the context of the web servers that respond to these URLs.

I consider the ocap properties with regards to two web servers:

- Waterken Server (`http://waterken.sourceforge.net`)
- PLT Web Server (`http://docs.racket-lang.org/web-server/`)

(When talking about PLT Web Server, I'll be referring to its stateful mode in this section, and I consider its stateless mode in section 3.)

Each of these servers provide an environment for managing pending computations and use different methods to map URLs to those computations. The delayed computation that these servers hold have different representations, with implications on how their respective servers hold onto them at runtime. These environments also use different encoding schemes to their respective URLs. To make the discussion comparing the two URL schemes of Waterken server and PLT Web server more precise, I'll distinguish between the two with the names:

- *web-key* for Waterken server.

  Although the exam question asks for y-urls, this is a type error: y-urls dictate how they're to be used to limit communicate to only those that deliberately share knowledge amongst each other [4], but offer no notion related to referring to a delayed computation. web-keys [2], on the other hand, act as a concrete instantiation of the y-url concept for the Waterken server, that connects URLs to server-side computations.

  A web-key is a URL scheme to represent an unforgeable, authenticated reference. It consists of three pieces.

  1. sha-1 fingerprint of the target's public key
  2. the network server
  3. a key to a computation

  The first two pieces of a web key are used to ensure that the target server that is being contacted is the intended party. Using the fingerprint to ensure the proper key exchange, the cryptographic handshake protocol completes, and then the third part of the URL is encrypted and sent over to complete the request, to support the unforgability and secrecy ocap properties.

- *k-url* for PLT Web server. A k-url consists of the host, a servlet instance identifier, and a continuation key. PLT Web server can support multiple servlets running at once, in which case the identifier helps to dispatch to the correct servlet. The continuation key is a reference to a computation.

2.3 Delayed computation and revokability (deliberate and inadvertent)

- Waterken server uses the notion of promises, which are essentially functions that are serialized on the server side into a database store. It uses Java's serialization mechanism to reify these promises to disk.
- PLT Web Server uses continuations to represent the pending computations. In stateful mode, these values can't be serialized, and are held in server-side memory.

Although promises and continuations are different in form, they both express delayed computation on the server side. Furthermore, because they depend so heavily on the server maintaining them on the server side, revokability can be implemented as an operation on the server side that deletes the association between URL and delayed computation. PLT Web server provides such a revocation operation in its `send/finish` primitive, and Waterken provides equivalent functionality.

One problematic effect of stateful PLT Web Server's storage of continuations in server-side memory is the use of limited server-side resources. PLT Web server can not represent an unbounded number of continuations in resident memory, and takes the unsound approach of expiring those continuations based on policies such as LRU. This means that certain messages can be inadvertently revoked under memory pressure. Essentially, a client with a k-url is holding a weak reference to a pending computation, and has no control over when that reference is garbage collected by the server. This behavior might not be correct for an ocap system.

PLT Web Server's behavior in stateless mode has a profound effect on ocap properties. We will revisit the question of revokability and the other ocap properties in the context of stateless mode in section 3.

2.4 Subdivisibility

Both of these servers allow delayed computations to be dynamically generated. A running computation can construct a new delayed computation, associate it with a URL, and have that delayed computation work with a more constricted set of operations. For example, a web-key or k-url can represent a computation with toplevel access to a user's entire social-networking profile, and the computation can spawn additional URLs associated to sub-computations that have limited, read-only access to that user's image resources.

This delegation can support subdivisibility as long as the newly-constructed computation elects to use fewer features than its origin. Some languages provide direct support to limit the power of delayed computations. Waterken server limits computation by using a object-capability language in Joe-E [10] which follows ocap principles. Racket, the language for PLT Web Server, can limit computations by running them within a sandboxed environment. These linguistic approaches help support subdivisibility.

2.5 Unforgability, leakage threats, and secrecy

With regards to ocap unforgability, both these servers assign a randomly generated key to a computation. Assuming a sufficiently long random key, and assuming the URL is kept from leaking unintentionally to other parties, the randomly-generated key enables the unforgability ocap property to hold because it's practically unguessable.

Leakage, however, is a complicated question in practice, particularly because web browsers perform operations that can leak these URLs to the world [2].

- One threat comes from a modern browser's anti-phishing support: some browsers enable spam or malware-detection by sending URLs to a central database to check against a blacklist. Neither web-keys nor k-urls currently defend against the leakage of URLs to such central authorities.

- Another of the threats identified by the Waterken authors is that of the referrer header field in an HTTP Request: if one visits a URL, and then goes from that to another page, the next page receives the URL of the origin.

k-urls do not defend against the leakage through the referrer header, so a browser can inadvertently leak k-urls to the outside world, hindering a k-urls use as a ocap. Web-keys can deal with the threat of leaking through the referrer header by keeping query information, including the random key, in the fragment segment of a URL. Browsers do not send the fragment portion within the referrer header over the network.

Recording key information within the fragment segment, as done with web-keys, has an even more drastic effect on the messaging protocol. Browsers do not send the fragment segment as part of the URL either, which means that a single HTTP request is not enough to complete a request with a web-key: the initial request doesn't include enough information to refer to the pending computation key. A web server that uses web-keys includes JavaScript code on the main page that performs a trampolined request with the `XMLHttpRequest` API:

1. The browser visits the toplevel page, which contains JavaScript code to start a trampoline.

2. The trampoline establishes a handshake with the server, using authentication information held in a portion of the web-key. (The fingerprint of the server's public key).

3. Once the authentication handshake is completed, the trampoline can perform a secure communication via public key encryption.

4. It sends the fragment content through XMLHttpRequest to complete the request.

Waterken's use of client-side JavaScript here protects the secret key from exposure. The use of the authentication handshake and public-key cryptography also allow web-keys to support both the unforgability and secrecy ocap properties. Although k-urls don't directly support the secrecy ocap property, they can use the `https` protocol to secure the connection.

3 PLT Web Server's Stateless Mode

PLT Web Server can also run in *stateless mode*, which offloads the representation of its continuations out of the server's memory into serializable structures. The essence of the approach is a combination of a program transformation to make continuations reifiable (either CPS [5] [8] or A-normal form [9]), along with lambda-lifting and defunctionalization to allow a continuation to be serialized as a byte stream.

PLT Web server can perform two fundamental things with a continuation byte stream to ensure it persists:

1. It can save a continuation to the server-side disk and associate it with a randomly generated key, or

2. It can encode the continuation directly in the continuation key portion of a k-url.

The first approach is analogous to that of Waterken's. Since continuations can live in long-term storage, and since disk storage is much cheaper than memory, there's no pressure to delete a serialized continuation, eliminating the threat of inadvertent revocation. Since these continuations are saved on the server side, deliberate revocation is an act of modifying the server-side values.

However, k-urls in stateless mode still have some weaknesses with regards to the ocap properties of unforgability and secrecy: they don't hide the secret key in a way that prevent leakage through the HTTP referrer header, and without using a protocol such as `https`, they don't maintain the secrecy of communication, which hurts their use in a distributed ocap setting.

3.1 Serialization into the URL

The second approach, to encode the continuation directly in the URL, allows the server to be purely stateless. The serialized continuation contains a representation of the individual continuation fragments, along with their associated environments. The k-url holds all the state of the computation, which allows the system to scale since the computation has no dependency to a particular server. However, this technique has major negative impact for ocap properties.

- If k-urls encoded in this way are evaluated without any restrictions, it allows the forgery of an arbitrary delayed computation.
- If continuations are in plain-text, it exposes the internal state of the delayed computation.
- Revocation has no meaning if the continuation is purely stateless.

At its essence, PLT Web Server's stateless mode takes a program and automatically constructs a specialized virtual machine whose atomic instructions are the continuation fragments of that program. In this view, a k-url encodes the code (references to the continuation frames) and data (contents of the lexical environment) segments of a program. One additional portion of a stateless k-url contains a MD5 hash of the program's source code, identifying the program with the particular virtual machine.

This twist of looking at PLT Web Server's stateless mode as the construction of a virtual machine evaluator, and of k-urls as programs for that evaluator, is crucial in identifying a critical ocap violation. If PLT Web Server allows k-urls to evaluate these continuations in an unrestricted way, and if the set of continuation fragments are rich enough, then this it allows arbitrary code evaluation: it allows forgery of a capability! It's akin to a stack overflow attack, except without the need to overflow. To prevent this violation, the server must ensure that the server is the only entity that can manufacture continuations. One simple way to do this is to add a signature to the k-url that verifies that its manufacturer is the server.

Exposing the continuation in the URL can leak the internal state of the delayed computation, in the form of the lexical environment of that computation. This, too, is a violation of object-capability systems because a transparent representation exposes information about objects that are supposed to be black boxes, and if the internal state of the continuation contains other k-urls, then a client can steal capabilities. To prevent this ocap violation, the server can encrypt the serialized continuation to make the representation opaque to prying eyes.

Finally, the idea of revocation hinges on the ability to mutate an object to no longer accept a message. If the continuation is purely stateless, revocation has no meaning because there's nothing to mutate on the server-side. McCarthy discusses this problem in the context of replay attacks [9]. To preserve the intentional revokability of k-urls, the delayed computation itself must depend on some mutable server state, such as a database, so that a computation can check that state before continuing a computation.

3.2 Serialized computations and code change

One other consideration that's related to revocation is that of code change: if the source code on the server side evolves, how do these code changes affect prior serialized computations? The permanence or transitory nature of URLs in the face of code change can affect their use as object capabilities in the real world.

One solution is to immediately revoke all existing URLs, which essentially happens when a PLT Web server in stateful mode has been stopped and restarted. Otherwise,

- In Waterken, since promises are serialized using the Java Serialization API [1], a Java programmer can preserve the `serialVersionUID` field of a promise class, effectively upgrading old computations to new versions.
- In PLT Web Server stateless mode, a change to a module's source code revokes those computations that are written in that module, because the md5 hash of the module's source has changed.

Web-keys, then, can offer more permanence than k-urls with regards to code change because the serialization mechanism in the Waterken server provides a documented mechanism for upgrading old computations to new ones directly. PLT Web Server doesn't provide linguistic support to upgrade serialized computations, so k-urls are more transient.

References

[1] Java object serialization specification. `http://download.oracle.com/javase/7/docs/platform/serialization/spec/serialTOC.html`.

[2] Tyler Close. Mashing with permission. `http://waterken.sourceforge.net/web-key/`.

[3] Tyler Close. Waterken server. `http://waterken.sourceforge.net`.

[4] Tyler Close. Waterken YURL: What does the 'y' refer to? `http://waterken.com/dev/YURL/Definition/#URL_f`.

[5] Paul Graunke, Robert Bruce Findler, Shriram Krishnamurthi, and Matthias Felliesen. Automatically Restructuring Programs for the Web. *Automated Software Engineering*, 2001.

[6] Carl Hewitt. Viewing Control Structures as Patterns of Passing Messages. Technical Report AIM-410, Massachusetts Institute of Technology Artificial Intelligence Laboratory, 1976.

[7] Shriram Krishnamurthi, Peter Walton Hopkins, Jay McCarthy, Paul T. Graunke, Greg Pettyjohn, and Matthias Felleisen. Implementation and Use of the PLT Scheme Web Server. *Higher Order and Symbolic Computation*, 2007.

[8] Jacob Matthews, Robert Bruce Findler, Paul Graunke, Shriram Krishnamurthi, and Matthias Felleisen. Automatically Restructuring Software for the Web. *Journal of Automated Software Engineering*, 2004.

[9] Jay McCarthy. Automatically RESTful Web Applications: Marking Modular Serializable Continuations. *Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming*, 2009.

[10] Adrian Mettler, David Wagner, and Tyler Close. Joe-E: A Security-Oriented Subset of Java. *Distributed Systems Symposium, Internet Society*, 2010.