Breaking the Ada Privacy Act

Jeffrey R. Carter
PragmAda Software Engineering
1540 Coat Ridge Road
Herndon, VA 22070-2728
72030.677@compuserve.com

The Ada Privacy Act of 1983

Ada 83 provides excellent protection for private types: Normal language features provide no way to access the underlying representation of a private type outside its package. Thus, the designer of a private type may safely make assumptions about the value of variables of a private type based solely on the operations on the type provided by the package. Since clients cannot "open the box" to access the full type definition, nor apply operations other than those provided, the designer can guarantee the behavior of the package.

There are always those who seek to circumvent such protection, out of concerns for efficiency, displeasure at being denied direct access, or the thrill of meeting a challenge. To ensure the functionality and reliability of our systems, we must be able to catch those who seek to break the Privacy Act. To do so, we must know their methods.

An Example

Consider a simple package for providing bounded, variable-length strings:

```
package String_Handler is
  type String_Handle is private; -- Initial value: the null string

  Too_Long : exception;
  -- Raised by To_Handle if the string will not fit in a String_Handle

  function To_Handle (Value : String) return String_Handle;
  -- Converts Value to a String_Handle
  -- Raises Too_Long if Value will not fit in a String_Handle

  function To_String (Handle : String_Handle) return String;
  -- Converts Handle to a String
  -- The lower index of the returned String is always 1

  function Length (Handle : String_Handle) return Natural;
  -- Returns the length of the string stored in Handle
private -- String_Handler
  type String_Handle is record
     Length : Natural := 0;
     Value : String (1 .. 252) := (others => ' ');
  end record;
end String_Handler;
```

While other operations may be desirable, this simple package is surprisingly useful as is, and is short enough to keep this article readable.

Now, suppose someone on our project has a `String_Handle` object, `S`, and needs to ensure that `Length (S) = N`. If `S` is too long, it should be truncated; if it is too short, it should be padded with `'*'`. The correct way to do this is:

```
if Length (S) > N then
   S := To_Handle (To_String (S) (1 .. N) );
else
   S := To_Handle (To_String (S) & (Length (S) + 1 .. N => '*') );
end if;
```

However, this strikes our developer as too much to type, and inefficient to boot. Also, the requirements state that the strings this code will process will never be too short. He wants to be able to write:

```
S.Length := N;
```

Breaking the Privacy Act with Unchecked Programming

One tool our developer has to achieve his ends is unchecked programming, specifically `Unchecked_Conversion`. Since he can read the private part of the package, he can duplicate the full type definition:

```
with Unchecked_Conversion;
...
  type My_Handle is record
      Length : Natural;
      Value : String (1 .. 252);
  end My_Handle;

  function Show is new Unchecked_Conversion (String_Handle, My_Handle);
  function Hide is new Unchecked_Conversion (My_Handle, String_Handle);

  LS : My_Handle := Show (S);
...
  LS.Length := N;
  S := Hide (LS);
```

Note that our developer has had to write nine statements to achieve his end, even though writing three statements to do it right was too much to type. However, he feels better about writing these statements because he knows they're more efficient.

This violation of the Privacy Act is easy to detect. We use a simple text-searching tool to detect all occurrences of "Unchecked_Conversion." We examine any we find, and reject any which are not acceptable. In properly designed systems, there should be few acceptable uses of `Unchecked_Conversion`.

Breaking the Privacy Act with System-Level Programming

We caught this violation of the Privacy Act, and sentenced the perpetrator to rewrite the offending code. He returns to his workstation, determined to outsmart us. In FORTRAN he could use an EQUIVALENCE; there must be an Ada equivalent.

```
...
  type My_Handle is record
      Length : Natural;
      Value : String (1 .. 252);
   end My_Handle;

   LS : My_Handle;
   for LS use at S'Address;
...
   LS.Length := N;
```

This is great! It only requires six statements, and is even more efficient, since it eliminates the assignment to `LS`. Best of all, our search for `Unchecked_Conversion` won't find it. Why didn't he think of this before?

Luckily for us, acceptable uses of address clauses in a properly designed system should be as rare as acceptable uses of `Unchecked_Conversion`. It is still fairly simple to search for "use at" and examine the results. While we're at it, we should search for "System" and "Address" as well.

Again we apprehend the miscreant and bring him to justice. He returns to work, but is unable to find another way around the Privacy Act. Defeated, he implements the length change correctly, vowing to get even some day.

The Ada Privacy Act of 1995

Ada 95 modifies the concepts of packages and private types in some interesting ways. Packages may have a hierarchy of child packages, which extend the capabilities of the package without changing its specification. Private types may be tagged, allowing clients to create new types by extending the type definition. A variation on private types allows private extension of tagged types.

Shortly after getting our developer to change string lengths correctly, we send a memo to all developers. We have obtained an Ada-95 compiler, and have decided to implement the system in Ada 95. All developers are encouraged to begin using the new features of the language immediately.

Ada 95 includes the standard package `Ada.Strings.Bounded`, which implements bounded, variable-length strings. We really should convert all uses of `String_Handler` to `Ada.Strings.Bounded`. However, enough code has already been written using `String_Handler` that we decide to retain it for now, rather that jeopardize our schedule. We'll convert the system to `Ada.Strings.Bounded` later. We send another memo instructing developers to use `Ada.Strings.Bounded` in all new code.

Breaking the Privacy Act with Child Packages

Our efficiency-minded developer has been browsing through the Ada-95 Reference Manual [1]. He notes that he can still break the privacy act with `Unchecked_Conversion` and with an address clause, and still get caught the same way. However, he is intrigued by the idea of child packages. Apparently, anyone can write a child package of any existing package, and the parent's private part is visible to the operations of a child package. He begins writing:

```
package String_Handler.Extensions is
  procedure Set_Length (Handle : in out String_Handle; To : in Natural);
  pragma Inline (Set_Length);
end String_Handler.Extensions;
```

```
package body String_Handler.Extensions is
  procedure Set_Length (Handle : in out String_Handle; To : in Natural) is
      -- null;
  begin -- Set_Length
      Handle.Length := To;
  end Set_Length;
end String_Handler.Extensions;

with String_Handler.Extensions;
...
  String_Handler.Extensions.Set_Length (Handle => S, To => N);
```

This required eight statements, although he could have left out the `pragma Inline` and made it seven. It's quite efficient, and does not involve unchecked or system-level programming. We do not detect this violation of the integrity of `String_Handle`, since it is not, technically, a violation of the Privacy Act.

A year later we deliver the system, and the developer leaves the company for a better-paying job, since he is now an experienced Ada-95 developer. Several months later, the users of our system begin to report apparently random failures.

Investigation determines that the failures are related to strings shorter than *N*, despite this not being required of the system. We assured our customer that we would handle short strings properly, even though this was not required, and our customer took us at our word. Finding the problem is a long and difficult process; our support contract does not begin to cover the costs involved.

Our code-inspection process did not find this problem: Inspectors must assume that packages, other than the code being inspected, are correct. `String_Handler.Extensions` was never inspected; our developer simply didn't bother to mention it.

What can we do to prevent this from recurring on future systems? Do we consider the use of child packages suspect, on a par with unchecked and system-level programming? Do we institute laborious, manual reviews of every child package mentioned in a context clause? Do we regret the day we decided to use Ada 95?

Reference

1. International Standards Organization, *Ada 95 Reference Manual,* ANSI/ISO/IEC 8652:1995