

Writing Secure Applications in Scheme

Win Treese

Serissa Research, Inc.

treese@serissa.com

13 October 2003

Creating Secure Applications

Security requirements

Design and architecture

Security and cryptographic components

Access management

Avoiding implementation flaws

Testing

Security Requirements

Confidentiality

Authentication

Authorization

Integrity

Availability

Privacy

Auditing

Robustness and reliability

What Security Problem?

Local applications, especially privileged ones

Clients executing downloaded (and possibly malicious) code

Servers and other applications under attack

What Security Problem?

Local applications, especially privileged ones

Clients executing downloaded (and possibly malicious) code

Servers and other applications under attack

How to break a Web application

Bypass its access controls

Break out of the application code

Make the application evaluate code improperly

Make the application invoke another interpreter improperly

Buffer overflows

In C:

```
int getinput (char * prompt )
{
    int status ;
    char input [BUFSIZ];
    printf ( "%s : _" , prompt );
    gets ( input );
    status = parse_input ( input );
    return status ;
}
```

Buffer overflows

Most common form of attack on the Internet today

Are Scheme application susceptible?

Is the underlying implementation safe?

One Partial Response: Java VM Byte Code Verifier

From *Inside Java 2 Platform Security* by Li Gong:

“Approximately, the code is checked to ensure the following.

- It does not forge pointers.
- It does not violate access restrictions [...]
- It accesses objects as what they are. [...]
- It calls methods with appropriate arguments of the appropriate types and there are no stack overflows.
- No illegal data conversions are done, such as converting integers to pointers.”

Web Applications: Dangerous Input

Sources: HTML forms, cookies, URLs

Hidden fields may be changed

Cookies may be modified

URLs can be edited

Referer may be forged

HTML input may be copied into other HTML pages

Example: SQL Injection

Example:

Input username from Web form

```
SELECT * FROM Users WHERE name=' $user '
```

Web: SQL Injection Attack

Suppose name is

```
' OR 1=1 ---
```

Then we get

```
SELECT * FROM Users WHERE name=' ' OR 1=1 ---
```

Web: Cross-Site Scripting

Consider:

```
http://www.example.com/search.pl?text=  
<script>alert(document.cookie)</script>
```

Scheme-Based Solutions

Easy to insert procedures to provide correct quoting

Example: `brl-sql-string`

Similar for quoting HTML, such as in BRL and SXML

But must still handle tampering

Dangerous characters

URL encodings

- In %xx form
- %00 \Rightarrow null byte
- %25 \Rightarrow %
- %2500 \Rightarrow %00 \Rightarrow null byte

Other issues in Unicode as well....

Foreign Function Interfaces

Interfaces to C may require explicit storage and buffer management

Inherits all the problems from the other side of the interface

Safety rules are likely different, and possibly incompatible

Eval

Is it dangerous?

Perl, Python, Tcl: `eval` called on strings

Scheme: `eval` called on expressions

A warning about strings

Alan Perlis:

“The string is a stark data structure and everywhere it is passed there is much duplication of process. It is a perfect vehicle for hiding information.”

Configuration files

May be tempted to use `read` and `eval` or `load`

Don't!

Usually easy to create a simple interpreter for what you need

But do not read and eval

Sandia experience: remember to

```
( setf *read-eval* nil )
```

Finer-grained control over read-time evaluation for Scheme is part of SRFI-10.

Denial of Service

Tie up input connections

Thrash under load

Force garbage collection or thrashing on data structures

Secure Execution Environments for Untrusted Code

Rees: “A Security Kernel Based on the Lambda-Calculus”

Sophisticated permissions management in Java

Sandboxing in PLT Scheme

Security Programming

Scheme has a relative lack of high-quality cryptographic libraries

Lack of protocol, encoding, and other critical libraries

Foreign interfaces particularly problematic

Protecting and destroying keys in memory is a problem

Need cryptographically strong random number generation

Does the language matter?

Buffer overflows a minor issue for Scheme, Perl, Python, Java, etc.

`eval` is an issue

Must be careful about particulars in any language

Culture and idioms probably matter as well

Is Scheme a good language for secure applications?

Yes:

- Core language is reasonably safe
- Easy to use constructs for input validation, etc.

But:

- Be careful with `eval` and friends
- Check all input
- Consider possible problems in the implementation
- Watch out for foreign function interfaces

The Scheme Security Project

Resources and guides for secure programming in Scheme

Resources for security programming

Discussion of security programming in Scheme

Making fun of security problems in other languages

Help find implementation problems and get them fixed

Common Lisp, too, if there's interest

Contact: Win Treese, treese@acm.org

Some Requests

Go home and check the input validation in your networked programs

If you teach programming:

- Make students aware of security issues
- Avoid using “obviously” unsafe programming examples
- Make fun of textbooks that use obviously unsafe examples

Contribute to the Scheme Security Project

Writing Secure Applications in Scheme

Win Treese

Serissa Research, Inc.

treese@serissa.com

13 October 2003