It All Starts Here...

I have a problem that is very common. Five years ago, developers at my company worked hard to design a secure product. But security requirements change so fast these days!

Our product is facing a lot of new attacks.

Application Developer
Moral of the Story

- New attacks are being launched all the time.
- Security is often an afterthought.

“(Security) cannot be added on, it must be designed from the beginning.” [A72]
Security Oriented Program Transformation

Security solutions can be described as program transformations.

Insecure software → Security Oriented Program Transformation → Protected Software

Defense in Depth

Partitioning → chroot Jail → Safe Library Replacement
## Buffer Overflow Attack

A buffer overflow, or buffer overrun, is a programming error which may result in a memory access exception and program termination, or in the event of the user being malicious, a breach of system security.

```c
/* overflow.c - demonstrates a buffer overflow */
#include <stdio.h>
#include <string.h>

int main (int argc, char *argv[]) {
    char buffer[12];
    if (argc < 2) {
        fprintf (stderr, "USAGE: %s string
", argv[0]);
        return 1;
    }
    strcpy (buffer, argv[1]);
    return 0;
}
```
1) **Source Code.** Replaces `strcat` and `strcpy` with `g_strlcat` and `g_strlcpy` from the glib library.
   
   **Example.** Replace,
   
   ```c
   strcpy (dst, src);
   ```
   
   with,
   
   ```c
   g_strlcpy (dst, src, sizeof(dst));
   ```

2) **Makefiles.** Include glib library during link time.
   
   **Example.** Replace,
   
   ```bash
   gcc
   ```
   
   with,
   
   ```bash
   gcc `pkg-config -libs glib-2.0`
   ```

3) **Configuration files.** Same as #2 in configuration files.
Evaluation

We have applied the program transformation to two open source programs with recent buffer overflow vulnerabilities. [Bugtraq 20978, Bugtraq 23013]

1) **Gnu Viewer.** 46 C files, 27000 LOC.
   - Replaced 86 of 88 instances in source code.
   - Replaced 4 of 4 instances in make and config files.
   - Buffer overflow vulnerability **absent** in the modified version.

2) **zzip lib library.** 33 C files, 7346 LOC.
   - Replaced 5 of 5 instances in source code.
   - Replaced 15 of 15 instances in make and config files.
   - Buffer overflow vulnerability **absent** in the modified version.

Safe Alternatives

<table>
<thead>
<tr>
<th>Unsafe Functions</th>
<th>Safe Alternatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>strcpy, strncpy</td>
<td>g_strcpy from glib library, astrcpy, astmncpy from libmib library, strncpy_s from ISO/IEC 24731</td>
</tr>
<tr>
<td>strcat, strncat</td>
<td>g_strcat from glib library, astrcat, astnccat from libmib library, strncat_s from ISO/IEC 24731</td>
</tr>
<tr>
<td>gets</td>
<td>gets_s from ISO/IEC 24731, fgets from C99, aflgets from libmib library</td>
</tr>
<tr>
<td>memcpy, memmove</td>
<td>memcpy_s from ISO/IEC 24731, memmove_s from ISO/IEC 24731</td>
</tr>
<tr>
<td>getenv</td>
<td>getenv_s function</td>
</tr>
<tr>
<td>SQL Statement</td>
<td>SQL PreparedStatement</td>
</tr>
<tr>
<td>mktemp</td>
<td>mktemp function</td>
</tr>
<tr>
<td>printf</td>
<td>snprintf function</td>
</tr>
</tbody>
</table>
SQL Injection Attack

An attacker inputs malicious SQL keywords

```java
import java.sql.*;

public class DBConnect {
  public void showData() {
    String username = stdin.readLine();
    try {
      String query = "select * from users" +
        " where username = '" + username + "');";
      preparedStatement = connection.prepareStatement(query);
      ResultSet resultSet = preparedStatement.executeQuery();
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }
}
```

Why do we make mistakes?

![Image of a cartoon character with a question mark]

© 1998 Randy Gasbergen.
**Decorated Filter Transformation**

Parameter inserted from variable var1 in program prog1

```
select * from table1 where param1 = ____ ;
```

SQL statement

![Diagram](image.png)

Input domain of variable var1 in program prog1

**Decorated Filter Transformation: Mechanism**

![Diagram](image.png)
import java.sql.*;
import model.PolicyDecorator;
import model.UnsafeString;
import model.SQLPolicy.SQLPolicyRemoveAnd;
import model.SQLPolicy.SQLPolicyRemoveOr;

public class DBConnect {
    ...
    public void showData() {
        ...
        UnsafeString username = new UnsafeString();
        try {
            username.setStr(stdin.readLine());
        } catch (IOException e1) {
        ...
        try {
            stmt = connection.createStatement();
            PolicyDecorator policy = new SQLPolicyRemoveAnd (new
            SQLPolicyRemoveOr (username));
            resultSet = stmt.executeQuery(
                "select * from users" +
                "where username = " +
                policy.convert().getStr() +
                ");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    ...
}
Security on Demand

Quick adoption of security solutions

Program transformation patch

Security patch

Ten Years from Now

My Advisors
Ralph Johnson, UIUC
Carl Gunter, UIUC
Sam Kamin, UIUC
Darko Marinov, UIUC
David Garlan, CMU

Application Programmer’s Bookshelf

Patterns Catalog: https://netfiles.uiuc.edu/mhafiz/www/sopt.pdf