

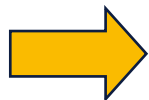
Embedded Testing 2017

Security by Static Source Code Analysis for Smaller Embedded Systems

Frank Büchner, Principal Engineer Software Quality, Hitex GmbH

- Since 1976 in Karlsruhe, Germany
- Approx. 40 employees
- Subsidiary in UK, approx. 20 employees
- Part of **Infineon**  group since 2003
- AURIX preferred design house (PDH)
- Tools for safety & security, middleware, compiler
- Engineering, production, consulting, test services





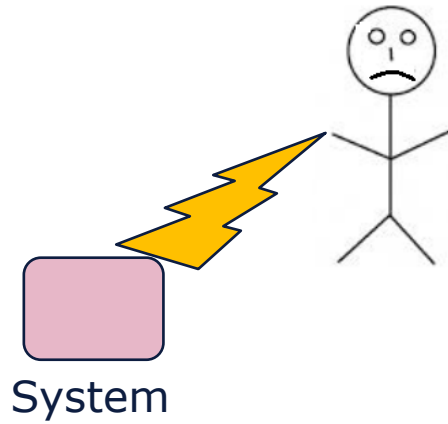
1. Safety & Security

2. Examples for Security Vulnerabilities

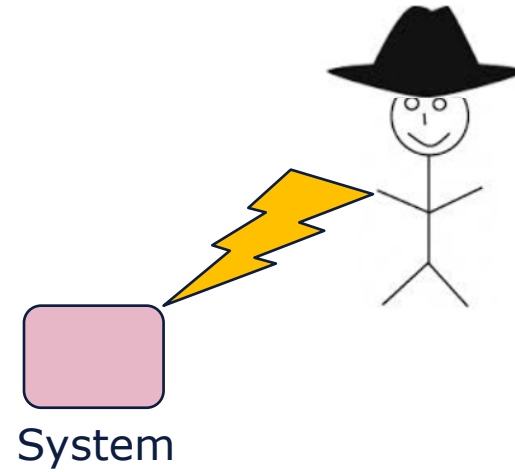
3. Security Vulnerability from the Standard C Library

4. Proprietary Security Vulnerability

Safety and Security

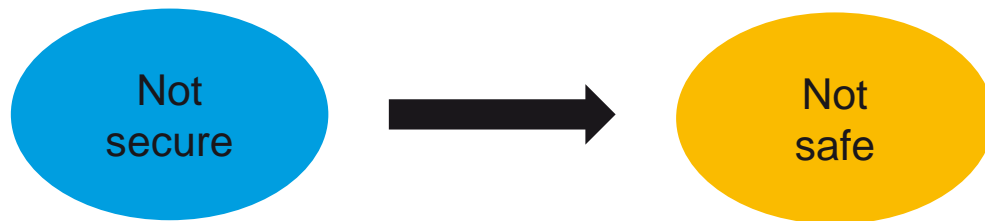


Safety

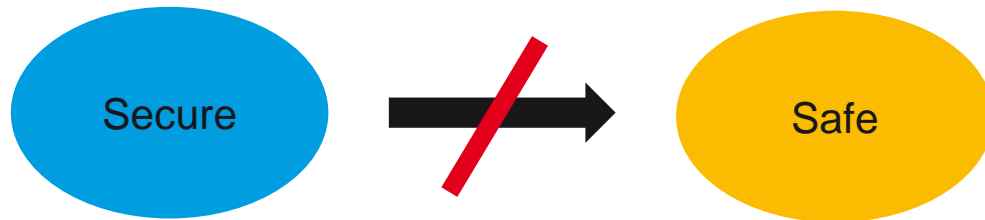


Security

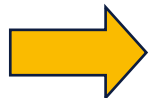
- If a system is not secure, it cannot be safe



- If a system is secure, it does not need to be safe



1. Safety & Security



2. Examples for Security Vulnerabilities

3. Security Vulnerability from the Standard C Library

4. Proprietary Security Vulnerability

■ Hardcoded passwords Sony IP Cam Backdoors

December 7, 2016 By Corey Nachreiner

Researchers found two **hard-coded accounts and passwords** in a large range of Sony's IP-based security cameras. Attackers with access to these cameras' web interface could use these credentials to take over the camera, even forcing access to a command line (CLI) interface. With botnets like Mirai actively looking for new victims, these sorts of IoT backdoors could put your IP-based cameras at risk. Watch our Daily Security Byte for the highlights, and if you own vulnerable cameras be sure to check out the firmware updates in the Reference section below.



Credit: [Sony Electronics](#)

Source: **Secplicity**
Security Simplified

■ Hardcoded passwords

□ CWE-259: Use of Hard-coded Password

```
int VerifyAdmin(char *password) {  
    if (strcmp(password, "Mew!")) {  
  
        printf("Incorrect Password!\n");  
        return(0)  
    }  
    printf("Entering Diagnostic Mode...\n");  
    return(1);  
}
```

- Unchanged default passwords
 - Linux.Muldrop.14
 - Attacks Raspberry Pi via port 22 (Secure Shell, SSH)
 - Uses the default password “raspberry” for the “pi” user
 - Mines for money and tries to find other Raspberry Pis

■ SQL Injection

School: Hi, this is your son's school. We're having some computer trouble.

Mom: Oh, dear -- Did he break something?

School: In a way. Did you really name your son `Robert'); DROP TABLE Students;--`?

Mom: Oh. Yes. Little Bobby Tables we call him.

School: Well, we've lost this year's student records. I hope you're happy.

Mom: And I hope you've learned to sanitize your database inputs.

bobby-tables.com

■ SQL Injection

□ Vulnerable code example

```
public static void DeleteUser(string username string connectionString)
{
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        string sqlQuery = String.Format("DELETE FROM Users WHERE UserName='{0}'", username);
        SqlCommand command = new SqlCommand(sqlQuery, connection);
        command.Connection.Open();
        command.ExecuteNonQuery();
    }
}
```

A SQL injection in this case could happen using the input “; DROP TABLE Users; --” that will delete all the users from the database instead of only deleting the users that have an exact name of *username*.

□ See also:

- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- Klocwork: CS.SQL.INJECT.LOCAL

■ SQL Injection

□ Fixed code example

```
public static void DeleteUser(string username, string connectionString)
{
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        string sqlQuery = "DELETE FROM Users WHERE UserName=@Username";
        SqlCommand command = new SqlCommand(sqlQuery, connection);
        command.Parameters.Add(new SqlParameter("@Username", username));
        command.Connection.Open();
        command.ExecuteNonQuery();
    }
}
```

A parameter is now used for the user name. A parameter will be treated as field data, not as part of the SQL statement.

■ XPath Injection

□ Vulnerable code example

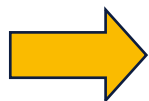
```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    super.doGet(request, response);
    String userName = request.getParameter("user.name");
    String userDescr = request.getParameter("user.descr");
    CompiledExpression compiledExpression =
        XPathContext.compile("string(//user[name/text()=' " + userName + "']/account/text())");
    compiledExpression.setValue(context, userDescr);
    response.getOutputStream().println("User info updated");
}
```

□ See also:

- CWE-643: Improper Neutralization of Data within XPath Expressions
- Klocwork: SV.XPATH

1. Safety & Security

2. Examples for Security Vulnerabilities



3. Security Vulnerability from the Standard C Library

4. Proprietary Security Vulnerability

■ Command injection

```
fgets(input_buf, sizeof(input_buf), fp);  
system(input_buf);
```

Vulnerability fgets()+system()

■ SV.TAINTED.INJECTION is raised

```
56  
57 fgets(input_buf, sizeof(input_buf), fp);  
58 system(input_buf);  
59
```

Klocwork Details

SV.TAINTED.INJECTION (Warning) [More information](#)

Unvalidated string 'input_buf' is received from an external function through a call to 'fgets' at line 57 that can be run as command line through call to 'system' at line 58. User input can be used to cause arbitrary command execution on the host system. Check strings for length and content when used for command execution.

Traceback:

- ▶ C:\Projects\Klocwork\Sanitize\Source\funcs.c:58
 - funcs.c:57: Tainted data 'input_buf' comes from call to 'fgets'.
 - funcs.c:58: Tainted data 'input_buf' is passed to 'system'.

■ SV.TAINTED.INJECTION is prevented

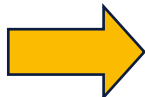
```
56  
57     fgets(input_buf, sizeof(input_buf), fp);  
58     strncpy(command, input_buf, BUF_LEN);  
59     sanitize(command);  
60     system(command);  
61
```

■ The input is sanitized

Whitelisting

```
void sanitize(char *command)
{
    char buf[BUF_LEN]="";

    if (0 == strncmp(command, "dir", 3))
    {
        strcpy(buf, "dir");
    }
    else if (0 == strncmp(command, "path", 4))
    {
        strcpy(buf, "path");
    }
    strncpy(command, buf, strlen(buf)+1);
}
```

1. Safety & Security
2. Examples for Security Vulnerabilities
3. Security Vulnerability from the Standard C Library
-  4. Proprietary Security Vulnerability

■ “Smaller” embedded systems

- Insulin pumps
- Heart pacemaker
- Smart electricity meters
- ...



■ Might not use Standard C library functions

■ No vulnerability is reported

```
113  
114     Read_command_from_Bluetooth(input_buf);  
115     Exec_command_from_Bluetooth(input_buf);  
116
```

- Remedy: Tell the static analysis tool about the vulnerability

```
Read_command_from_Bluetooth - TSSrc $1  
Exec_command_from_Bluetooth - TSSinkInj $1
```

- With the extended knowledge base, the vulnerability is reported

```
113  
114     Read_command_from_Bluetooth(input_buf);  
115     Exec_command_from_Bluetooth(input_buf);  
116
```

Klocwork Details

SV.TAINTED.INJECTION (Warning) [More information](#)

Unvalidated string 'input_buf' is received from an external function through a call to 'Read_command_from_Bluetooth' at line 114 that can be run as command line through call to 'Exec_command_from_Bluetooth' at line 115. User input can be used to cause arbitrary command execution on the host system. Check strings for length and content when used for command execution.

Traceback:

- ▲ C:\Projects\Klocwork\Sanitize\Source\funcs.c:115
 - funcs.c:114: Tainted data 'input_buf' comes from call to 'Read_command_from_Bluetooth'.
 - funcs.c:115: Tainted data 'input_buf' is passed to 'Exec_command_from_Bluetooth'.

- SV.TAINTED.INJECTION is prevented through sanitizing

```
113  
114     Read_command_from_Bluetooth(input_buf);  
115     strncpy(command, input_buf, BUF_LEN);  
116     Sanitize_command_from_Bluetooth(command);  
117     Exec_command_from_Bluetooth(command);  
118
```

Thank you !



Any questions?



Static source code analysis tool KLOCWORK
www.hitex.de/klocwork

Contact & Additional Information

Frank Büchner
Dipl.-Inform.
Principal Engineer Software Quality
Hitex GmbH
Greschbachstr. 12
Karlsruhe
Germany



Tel.: +49 / 721 / 9628 – 125
frank.buechner@hitex.de

Additional information:
www.hitex.de/klocwork
www.hitex.com/klocwork