Security Bugs Fixing Seminar Kick-Off

Technical University Munich
Department of Informatics
Chair for IT Security (I20)
Prof. Dr. Claudia Eckert

Paul Muntean
July 9, 2015
1. Definitions
2. Examples
3. Security Bugs Fixing
4. Tool Demo
5. Take Away!
Definition 1: Software Defect (Bug)

A software bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

Definition 2: Software Vulnerability

A software vulnerability is a security flaw, glitch, or weakness found in software or in an operating system (OS) that can lead to security concerns.

Definition 3: Defect Repair (Bug Fixing)

A change to a program or system intended to permanently remove a bug without introducing new bugs.
Definition 1: Software Defect (Bug)
A software bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.
Definitions

Definition 1: Software Defect (Bug)
A software bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

Definition 2: Software Vulnerability
Definitions

Definition 1: Software Defect (Bug)
A software bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

Definition 2: Software Vulnerability
A software vulnerability is a security flaw, glitch, or weakness found in software or in an operating system (OS) that can lead to security concerns.
Definitions

**Definition 1: Software Defect (Bug)**
A software bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

**Definition 2: Software Vulnerability**
A software vulnerability is a security flaw, glitch, or weakness found in software or in an operating system (OS) that can lead to security concerns.

**Definition 3: Defect Repair (Bug Fixing)**
Definitions

Definition 1: Software Defect (Bug)
A software bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

Definition 2: Software Vulnerability
A software vulnerability is a security flaw, glitch, or weakness found in software or in an operating system (OS) that can lead to security concerns.

Definition 3: Defect Repair (Bug Fixing)
A change to a program or system intended to permanently remove a bug without introducing new bugs.
<table>
<thead>
<tr>
<th>1</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Examples</td>
</tr>
<tr>
<td>3</td>
<td>Security Bugs Fixing</td>
</tr>
<tr>
<td>4</td>
<td>Tool Demo</td>
</tr>
<tr>
<td>5</td>
<td>Take Away!</td>
</tr>
</tbody>
</table>
## CWE/SANS Top 25 Most Dangerous Software Errors

<table>
<thead>
<tr>
<th>Rank</th>
<th>Score</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>93.8</td>
<td>CWE-89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>2</td>
<td>83.3</td>
<td>CWE-78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS C. Injection')</td>
</tr>
<tr>
<td>3</td>
<td>79.0</td>
<td>CWE-120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>4</td>
<td>77.7</td>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>5</td>
<td>76.9</td>
<td>CWE-306</td>
<td>Missing Authentication for Critical Function</td>
</tr>
<tr>
<td>6</td>
<td>76.8</td>
<td>CWE-862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>7</td>
<td>75.0</td>
<td>CWE-798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>8</td>
<td>75.0</td>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>9</td>
<td>74.0</td>
<td>CWE-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>10</td>
<td>73.8</td>
<td>CWE-807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>11</td>
<td>73.1</td>
<td>CWE-250</td>
<td>Execution with Unnecessary Privileges</td>
</tr>
<tr>
<td>12</td>
<td>70.1</td>
<td>CWE-352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>13</td>
<td>69.3</td>
<td>CWE-22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td>14</td>
<td>68.5</td>
<td>CWE-494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>15</td>
<td>67.8</td>
<td>CWE-863</td>
<td>Incorrect Authorization</td>
</tr>
<tr>
<td>16</td>
<td>66.0</td>
<td>CWE-829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
</tr>
<tr>
<td>17</td>
<td>65.5</td>
<td>CWE-732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
</tr>
<tr>
<td>18</td>
<td>64.6</td>
<td>CWE-676</td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td>19</td>
<td>64.1</td>
<td>CWE-327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td>20</td>
<td>62.4</td>
<td>CWE-131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td>21</td>
<td>61.5</td>
<td>CWE-307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
</tr>
<tr>
<td>22</td>
<td>61.1</td>
<td>CWE-601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
</tr>
<tr>
<td>23</td>
<td>61.0</td>
<td>CWE-134</td>
<td>Uncontrolled Format String</td>
</tr>
<tr>
<td>24</td>
<td>60.3</td>
<td>CWE-190</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>25</td>
<td>59.9</td>
<td>CWE-759</td>
<td>Use of a One-Way Hash without a Salt</td>
</tr>
</tbody>
</table>
## CWE/SANS Top 25 Most Dangerous Software Errors

<table>
<thead>
<tr>
<th>Rank</th>
<th>Score</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>93.8</td>
<td>CWE-89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Inject')</td>
</tr>
<tr>
<td>2</td>
<td>83.3</td>
<td>CWE-78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS C. Inject')</td>
</tr>
<tr>
<td>3</td>
<td>79.0</td>
<td>CWE-120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>4</td>
<td>77.7</td>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>5</td>
<td>76.9</td>
<td>CWE-306</td>
<td>Missing Authentication for Critical Function</td>
</tr>
<tr>
<td>6</td>
<td>76.8</td>
<td>CWE-862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>7</td>
<td>75.0</td>
<td>CWE-798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>8</td>
<td>75.0</td>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>9</td>
<td>74.0</td>
<td>CWE-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>10</td>
<td>73.8</td>
<td>CWE-807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>11</td>
<td>73.1</td>
<td>CWE-250</td>
<td>Execution with Unnecessary Privileges</td>
</tr>
<tr>
<td>12</td>
<td>70.1</td>
<td>CWE-352</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>13</td>
<td>69.3</td>
<td>CWE-22</td>
<td>Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')</td>
</tr>
<tr>
<td>14</td>
<td>68.5</td>
<td>CWE-494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>15</td>
<td>67.8</td>
<td>CWE-863</td>
<td>Incorrect Authorization</td>
</tr>
<tr>
<td>16</td>
<td>66.0</td>
<td>CWE-829</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
</tr>
<tr>
<td>17</td>
<td>65.5</td>
<td>CWE-732</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
</tr>
<tr>
<td>18</td>
<td>64.6</td>
<td>CWE-676</td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td>19</td>
<td>64.1</td>
<td>CWE-327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td>20</td>
<td>62.4</td>
<td>CWE-131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td>21</td>
<td>61.5</td>
<td>CWE-307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
</tr>
<tr>
<td>22</td>
<td>61.1</td>
<td>CWE-601</td>
<td>URL Redirection to Untrusted Site ('Open Redirect')</td>
</tr>
<tr>
<td>23</td>
<td>61.0</td>
<td>CWE-134</td>
<td>Uncontrolled Format String</td>
</tr>
<tr>
<td>24</td>
<td>60.3</td>
<td>CWE-190</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>25</td>
<td>59.9</td>
<td>CWE-759</td>
<td>Use of a One-Way Hash without a Salt</td>
</tr>
</tbody>
</table>
SQL Inject

**Figure:** SQL injection

```sql
SELECT * FROM `users`
WHERE `username` = 'johnsmith'
AND `password` = 'mypassword'

SELECT * FROM `users`
WHERE `username` = '' OR 1 = 1; /*
AND `password` = "*/
```
OS Command Injection

URL: localhost/vultest/lame.php?cmd=ping -c 3 exploitsecurity.blogspot.com
OS Command Injection

URL: localhost/vul.test/lame.php?cmd=ping -c 3 exploitsecurity.blogspot.com

We will test OS command injection vulnerability against this pages. Actually developer don’t know how serious the code is.

Output of command:

PING blogspot.l.google.com (209.85.175.132) 56(84) bytes of data, 64 bytes from nx-in-f132.1e100.net (209.85.175.132): icmp_seq=1 ttl=52 time=128 ms 64 bytes from nx-in-f132.1e100.net (209.85.175.132): icmp_seq=2 ttl=52 time=154 ms 64 bytes from nx-in-f132.1e100.net (209.85.175.132): icmp_seq=3 ttl=52 time=157 ms — blogspot.l.google.com ping statistics — 3 packets transmitted, 3 received, 0% packet loss, time 2001ms rtt min/avg/max/mdev = 128.959/146.916/157.752/12.786 ms

This is how OS command injection vulnerability works.

Figure: OS Command Injection Scenario
Buffer Overflow

```
1 #define BUFSIZE 256
2 int main(int argc, char **argv) {
3 char buf[BUFSIZE];
4 strcpy(buf, argv[1]);
5 }
```
Cross-site Scripting (XSS)

Figure: XSS Scenario
Missing Authentication for Critical Function I

Listing 1: Java Code Snippet (missing authentication)

```java
public BankAccount createBankAccount(String accountNumber, String accountType,
                                       String accountName, String accountSSN, double balance) {
    BankAccount account = new BankAccount();
    account.setAccountNumber(accountNumber);
    account.setAccountType(accountType);
    account.setAccountOwnerName(accountName);
    account.setAccountOwnerSSN(accountSSN);
    account.setBalance(balance);
    return account;
}
```
Missing Authentication for Critical Function II

Listing 2: Fixed Version

```java
private boolean isUserAuthentic = false;

// authenticate user,
// if user is authenticated then set variable to true
// otherwise set variable to false
public boolean authenticateUser(String username, String password) {
    ...
}

public BankAccount createNewBankAccount(String accountNumber, String accountType,
    String accountName, String accountSSN, double balance) {
    BankAccount account = null;

    if (isUserAuthentic) {
        account = new BankAccount();
        account.setAccountNumber(accountNumber);
        account.setAccountType(accountType);
        account.setAccountOwnerName(accountName);
        account.setAccountOwnerSSN(accountSSN);
        account.setBalance(balance);
    }
    return account;
}
```
1. Definitions

2. Examples

3. Security Bugs Fixing

4. Tool Demo

5. Take Away!
Bug Detection and Fixing Steps
Bug Detection and Fixing Steps

- Detect the bug
Bug Detection and Fixing Steps

- Detect the bug (Which file and line number?)
Bug Detection and Fixing Steps

- Detect the bug (Which file and line number?)
- Classify the bug
Bug Detection and Fixing Steps

- Detect the bug (Which file and line number?)
- Classify the bug (Which type of bug is it?)
Bug Detection and Fixing Steps

- Detect the bug (Which file and line number?)
- Classify the bug (Which type of bug is it?)
- Infer bug fix
Bug Detection and Fixing Steps

- Detect the bug (Which file and line number?)
- Classify the bug (Which type of bug is it?)
- Infer bug fix (What should be done to remove it?)
Bug Detection and Fixing Steps

- Detect the bug (Which file and line number?)
- Classify the bug (Which type of bug is it?)
- Infer bug fix (What should be done to remove it?)
- Determine insertion location
Bug Detection and Fixing Steps

- Detect the bug (Which file and line number?)
- Classify the bug (Which type of bug is it?)
- Infer bug fix (What should be done to remove it?)
- Determine insertion location (Where should I fix?)
Information Exposure Bugs

A definition from http://cwe.mitre.org/

**CWE-200:** An information exposure is the intentional or unintentional disclosure of information to an actor that is not explicitly authorized to have access to that information.
Information Exposure Bugs

A definition from http://cwe.mitre.org/

**CWE-200:** An information exposure is the intentional or unintentional disclosure of information to an actor that is not explicitly authorized to have access to that information.

**Other examples from Mitre:**
**CWE-526** (Info. Exposure Through Environmental Variables)
Information Exposure Bugs

A definition from http://cwe.mitre.org/

**CWE-200**: An information exposure is the intentional or unintentional disclosure of information to an actor that is not explicitly authorized to have access to that information.

Other examples from Mitre:

**CWE-526** (Info. Exposure Through Environmental Variables)

Info. exposure bugs are a subcategory of information flow bugs
Example Scenario with Code from CWE-526

```c
void CWE526_bad(){
    if (staticFive == 5){
        /* FLAW: environment variable exposed */
        printLine(getenv("PATH"));
    }
}
```

Example Scenario with Code from CWE-526

```c
void CWE526_bad(){
    if (staticFive == 5){
        /* FLAW: environment variable exposed */
        printLine(getenv("PATH"));
    }
}

void printLine (const char *line){
    if(line != NULL){
        printf("%s\n", line);
    }
}
```
Example Scenario with Code from CWE-526

1  void CWE526_bad(){
2     if (staticFive == 5){
3        /* FLAW: environment variable exposed */
4        printLine(getenv("PATH"));
5     }
6 }

1  void printLine (const char *line){
2     if(line != NULL){
3        printf("%s\n", line);
4     }
5 }

-> /sbin:/usr/sbin:/bin:/usr/bin:/usr/local/bin
CWE-526 Bug Fix

```c
1 void CWE526_good(){
2     if (staticFive == 5){
3         /* FIX: error message is general */
4         printLine("Not in path");
5     }
6 }
```
Test suites + Genetic programming

GenProg IEEE TRANSACTIONS ON SE 2011 [1], ICSE 2009 [2]
- Test suites used to encode the defect and required functionality
- Uses an extended form of genetic programming to evolve a program variant that retains required functionality but is not susceptible to a given defect
- Needs the correct statement somewhere in the code
- Security bugs addressed
  - nullhttpd, lighttpd: Remote Heap Buffer Overflow
  - openldap: Nonoverflow Denial of Service
  - php: Integer Overflow, etc.

- Pros: Widely used YouTube, No Formal Specifications No Program Annotations
- Cons: Special Crafted Test Suites
Test suites + SMT solving


- Test Cases (TC) used to tell if the program behaviour is correct or not, **Fault isolation**: Uses the ranking produced by a statistical fault isolation tool, **Statement-level specification inference**: discover the correct specification of the buggy statement, **Program synthesis**: synthesize an expression that conforms to the specification discovered before, SMT solving used to solve a repair constraints

- Bug types same as GenProg (GP)
- Pros: Almost 2x faster than GP, More bugs detected than GP
- Cons: Fault localization is based on statement suspiciousness rate. A statement exercised by more failing tests and fewer passing tests will have a higher suspiciousness score, special crafted TC needed
Free form bug reports

R2Fix [4] ICST-2013
- Generate bug-fixing patches from free-form bug reports
- Uses past fix patterns
- Machine learning techniques
- Semantic patch generation techniques
- Security bugs addressed in Linux kernel, Mozilla, and Apache
  - buffer overflows
  - null pointer bugs
  - memory leaks

Pros: Scales for Large Projects (0.3 M - 11.9 M), C/C++
Cons: uses fix patterns, needs past developer patches, manually reading of closed bug reports, manually labelling bug reports, patches are correct they are identical or semantically equivalent to the developer generated patch
Patch patterns + SMT solving

- Uses statically defined patches
- Uses a buffer overflow checker to localize the bug
- Solves a SMT system
- Inserts generated values inside patch pattern
- Insert bug fix at root cause location or at the bug location
- Security bugs addressed in CWE-121
  - stack based buffer overflows quick fixes

Pros: Precise bug detection, Precise quick fix insertion location, Automated patch correctness check, Semi-Automated patch insertion mechanism
Cons: Does not scale to large projects, Needs a database of predefined patch patterns
1. Definitions
2. Examples
3. Security Bugs Fixing
4. Tool Demo
5. Take Away!
Tool Demo
1 Definitions

2 Examples

3 Security Bugs Fixing

4 Tool Demo

5 Take Away!
Recall: Bug Detection and Fixing Steps

**Security Bug Fixing (SBF):**
- Correct bug localization
- Bug classification
- Fix generation
- Fix insertion location
- Program behaviour preserving

It is based on:
- Test Cases
- SMT solving
- Symbolic execution
- Patch patterns
- Machine learning
- Code synthesis
- Free form bug reports
- Reusage of human written patches

What other sources of information can be leveraged and/or combined in order to improve SBF?
Recall: Bug Detection and Fixing Steps

Four Questions: Which file and line number? Which type of bug is it? What should be done to remove it? Where should I fix?
Recall: Bug Detection and Fixing Steps

Four Questions: Which file and line number? Which type of bug is it? What should be done to remove it? Where should I fix?

- Security Bug Fixing (SBF) is about
Recall: Bug Detection and Fixing Steps

Four Questions: Which file and line number? Which type of bug is it? What should be done to remove it? Where should I fix?

- Security Bug Fixing (SBF) is about
  - Correct bug localization

It is based on: Test Cases, SMT solving, Symbolic execution, Patch patterns, Machine learning, Code synthesis, Free form bug reports, Reusage of human written patches, etc.

What other sources of information can be leveraged and/or combined in order to improve SBF?
Recall: Bug Detection and Fixing Steps

Four Questions: Which file and line number? Which type of bug is it? What should be done to remove it? Where should I fix?

- Security Bug Fixing (SBF) is about
  - Correct bug localization
  - Bug classification
Recall: Bug Detection and Fixing Steps

Four Questions: Which file and line number? Which type of bug is it? What should be done to remove it? Where should I fix?

- Security Bug Fixing (SBF) is about
  - Correct bug localization
  - Bug classification
  - Fix generation
Recall: Bug Detection and Fixing Steps

Four Questions: Which file and line number? Which type of bug is it? What should be done to remove it? Where should I fix?

- Security Bug Fixing (SBF) is about
  - Correct bug localization
  - Bug classification
  - Fix generation
  - Fix insertion location

It is based on: Test Cases, SMT solving, Symbolic execution, Patch patterns, Machine learning, Code synthesis, Free form bug reports, Reusage of human written patches, etc.

What other sources of information can be leveraged and/or combined in order to improve SBF?
Recall: Bug Detection and Fixing Steps

Four Questions: Which file and line number? Which type of bug is it? What should be done to remove it? Where should I fix?

- Security Bug Fixing (SBF) is about
  - Correct bug localization
  - Bug classification
  - Fix generation
  - Fix insertion location
  - Program behaviour preserving

No other bugs are introduced and the bug is really removed, check it!

It is based on: Test Cases, SMT solving, Symbolic execution, Patch patterns, Machine learning, Code synthesis, Free form bug reports, Reusage of human written patches, etc.

What other sources of information can be leveraged and/or combined in order to improve SBF?
Recall: Bug Detection and Fixing Steps

Four Questions: Which file and line number? Which type of bug is it? What should be done to remove it? Where should I fix?

- Security Bug Fixing (SBF) is about
  - Correct bug localization
  - Bug classification
  - Fix generation
  - Fix insertion location
  - Program behaviour preserving
  - No other bugs are introduced and the bug is really removed, check it!
Recall: Bug Detection and Fixing Steps

Four Questions: Which file and line number? Which type of bug is it? What should be done to remove it? Where should I fix?

- Security Bug Fixing (SBF) is about
  - Correct bug localization
  - Bug classification
  - Fix generation
  - Fix insertion location
  - Program behaviour preserving
  - No other bugs are introduced and the bug is really removed, check it!

- It is based on: Test Cases, SMT solving, Symbolic execution, Patch patterns, Machine learning, Code synthesis, Free form bug reports, Reusage of human written patches, etc.
Recall: Bug Detection and Fixing Steps

Four Questions: Which file and line number? Which type of bug is it? What should be done to remove it? Where should I fix?

- Security Bug Fixing (SBF) is about
  - Correct bug localization
  - Bug classification
  - Fix generation
  - Fix insertion location
  - Program behaviour preserving
  - No other bugs are introduced and the bug is really removed, check it!

- It is based on: Test Cases, SMT solving, Symbolic execution, Patch patterns, Machine learning, Code synthesis, Free form bug reports, Reusage of human written patches, etc.

- What other sources of information can be leveraged and/or combined in order to improve SBF?
Managerial Notes

- Start from 19.10.2015
- Every Monday 17:00 - 18:30 PM
- Room 01.08.033
- Presentation and Report
- Report (maximum 10 pages) is due to 25.01.16
- Presentation 30 min. & 5 min. Q & A
- Choose your topic by mailing me
Topic Selection

- 8 papers in total
- Choose 1 of them with priority before 19.09.15
- Send me an E-Mail with a short motivation letter (Seminar web page)
- Your topic will be announced on 30.09.15
- Cancellation only via mail before 10.09.15
Contact

- http://www.sec.in.tum.de/security-bugs-fixing/
- Paul Muntean, E-Mail: paul@sec.in.tum.de
- Meetings & advices on demand
<table>
<thead>
<tr>
<th>Definitions</th>
<th>Examples</th>
<th>Security Bugs Fixing</th>
<th>Tool Demo</th>
<th>Take Away!</th>
</tr>
</thead>
</table>

**Q & A**

Security Bugs Fixing Seminar Kick-Off
References


