SWE 781
Secure Software Design and Programming

Error Handling
Lecture 5

IATAC

Ron Ritchey, Ph.D.
Chief Scientist
703/377.6704
Ritchey_ronald@bah.com

Schedule (tentative)

Date	Subject
Sep 1st	Introduction (today) ; Chess/West chapter 1, Wheeler chapters 1,2,3
Sep 8th	Computer attack overview
Sep 15th	Input Validation; Chess/West chapter 5, Wheeler chapter 5
Sep 22nd	Buffer Overflows; Chess/West chapters 6, 7; Wheeler chapter 6
Sep 29th	Class Cancelled
Oct 6th	**Error Handling; Chess/West chapter 8; Wheeler chapter 9 (9.1, 9.2, 9.3 only)**
Oct 13th	Columbus Recess
Oct 20th	Mid-Term exam
Oct 27th	Mid Term Review / Major Assignment Introduction; Privacy, Secrets, and Cryptography; Chess/West chapter 11; Wheeler chapter 11 (11.3, 11.4, 11.5 only)
Nov 3rd	Implementing authentication and access control
Nov 10th	Web Application Vulnerabilities; Chess/West chapter 9,10
Nov 17th	Secure programming best practices / Major Assignment Stage Check ; Chess/West chapter 12; Wheeler chapters 7,8,9,10
Nov 24th	Static Code Analysis & Runtime Analysis
Dec 1st	The State of the Art (guest lecturer)
Dec 8th	TBD (Virtual Machines, Usability [phishing], E-Voting, Privilege Separation, Java Security, Network Security & Worms)

IATAC

Today's Agenda *

- Error Handling, What could possibly go wrong?
- Handling return codes
- Managing exceptions
- Preventing resource leaks
- Logging and debugging
- Minor Assignment 3

* Today's materials derive heavily from Chess/West, Securing Programming with Static Analysis

IATAC

Error Handling: What could possibly go wrong?

Ariane Flight 501

4 June 1996

IATAC

Ariane V crashed due to an unhandled exception

- Error was an arithmetic overflow caused by the conversation from a 64 bit float into a 16 bit integer value
- Value recorded horizontal velocity
- Software based upon software from the Ariane IV
  - A much slower rocket with a very different launch profile
  - The Ariane V launches much faster and more vertically
- Even though the rocket had redundant CPUs this didn't help because both were running the exact same software
- Routine that failed part of a realignment routine that ***was not needed*** on the Ariane V

IATAC

Today's Agenda

- Error Handling, What could possibly go wrong?
- Handling return codes
- Managing exceptions
- Preventing resource leaks
- Logging and debugging
- Minor Assignment 3

IATAC

Returns often overloaded to include return value and error codes

- Technique very popular with C/ C++ code but shows up in most languages
- Forces caller to differentiate between valid and error
- No common semantics are enforced. Every programming can implement the return/error logic differently
- Easy to ignore errors

```
#define MAXVALUE 1023
#define ERR_NULL -1
#define ERR_TOOBIG -2

int my_strlen(char *s) {
  int c=0; i;

  if (s == NULL) return ERR_NULL;
  for (i = 0; i < MAXVALUE; i++) {
    if (s[i] == '\0') return c;
    c++;
  }
  return ERR_TOOBIG;
}
```

IATAC

Return value error codes often ignored

- Very common error to not validate return values especially for functions that do not normally experience error conditions
- Creates many exploitable vulnerabilities
  - Remember normal is not what the attackers will provide

Ignored return creates buffer overflow

```
char buf[10], cp_buf[10];
fgets(buf, 10, stdin);
strcpy(cp_buf, buf);
```

Correct version eliminates overflow

```
char buf[10], cp_buf[10];
char * ret = fgets(buf, 10, stdin);
if (ret != buf) {
  report_error(errno);
  return;
}
strcpy(cp_buf, buf);
```

IATAC

Burying error values in same "space" as valid values requires calling function to parse the difference

- There is only a single return value from a function
  - Valid vs. Error must be determined by how an error is encoded into the return value
- Different functions use different methods for doing this
  - Might use < 0 to indicate error in numeric returns
  - Might use 0 to indicate error with global variable used to indicate type of error
  - Might screw it up and include a valid value as an error value (e.g. 0 might be a correct value or indicate function failure)

What's wrong with this code?

```
#define MAXVALUE 1023
#define ERR 0

int my_strlen(char *s) {
  int c=0; i;

  if (s == NULL) return ERR;
  for (i = 0; i < MAXVALUE; i++) {
    if (s[i] == '\0') return c;
    c++;
  }
  return ERR;
}
```

IATAC

Error handling interspersed with functional logic makes it easier to make mistakes

- Anything that increases complexity increases bugs
- Common mistakes from mixing error handling into function code include:
  - Not catching all error conditions
  - Changing the functional intention of the code when errors occur
  - Not cleaning up especially where dynamic memory management is concerned

```
#define MAXVALUE 1023
#define ERR_NULL -1
#define ERR_TOOBIG -2

char * func(char *s) {
  int c=0; i;
  char * sl;

  sl = malloc(sizeof(char)*MAXVALUE);
  if (s == NULL || sl == NULL)
    return (char *) ERR_NULL;

  for (i = 0; i < MAXVALUE; i++) {
    sl[i] = s[i];
    if (s[i] == '\0') return sl;
  }

  free(sl);
  return (char *) ERR_TOOBIG;
}
```

IATAC

Better to push error handling to the end of the function so recovery activities are consistent

- Separating error handling logic from functional logic makes the code easier to read and more maintainable
- Makes recovery activities consistent.
  - Important for any state changes that need to be rolled back e.g.
    - Dynamic memory allocation
    - Resource locking

```
char * func(char *s) {
  int c=0,i,err_val=0;
  char * sl=NULL;

  sl = malloc(sizeof(char)*MAXVALUE);
  if (s == NULL || sl == NULL) {
    err_val = ERR_NULL;
    goto ERR_HANDLER;
  }

  for (i = 0; i < MAXVALUE; i++) {
    sl[i] = s[i];
    if (s[i] == '\0') return sl;
  }
  err_val = ERR_TOOBIG;

ERR_HANDLER:
  if (sl != NULL) free(sl);
  return err_val;
}
```

IATAC

Today's Agenda

- Error Handling, What could possibly go wrong?
- Handling return codes
- Managing exceptions
- Preventing resource leaks
- Logging and debugging
- Minor Assignment 3

IATAC

Exceptions more elegant system for managing error handling

- Exceptions allows structure to formally separate functional logic from error condition handling
  - Uses a try/catch structure

```
try {
  some code that could produce exception
}
catch (BadException e) {
   log it and recover
}
```

- Can be checked or unchecked
  - Java compiler verifies that exceptions are addressed in code
  - C++ does not have this feature
  - Checked exceptions force developers to specifically handle error conditions

IATAC

Unhandled exceptions will cause program termination; this is a good thing

- Important to differentiate between exceptions that can be handled safely within the program and those that require the process to exit
- Need to make sure that exception data is not communicated to the user of the application
  - Requires top level exception handler to catch remaining exceptions before terminating the application

```
protected void doPost (HttpServleyRequest req, HttpServletResponse res) {
  try {
    String ip = req.getRemoteAddr();
    InetAddress addr = InetAddress.getByName(ip);
    out.println("hello " + Utils.processHost(addr.getHostName()));
  }
  catch (UnknownHostException e) { logger.error("ip lookup failed", e); }
  catch (Throwable t) {
   logger.error("caught Throwable at top level", t); }
}
```

Network servers (e.g. Web) in particular should not allow users to see unhandled exception data

Application: photosprintshopWeb
Error: java.lang.IllegalStateException exception
Reason:
java.lang.IllegalStateException: An Exception occurred while generating the Exception page 'WOExceptionPage'. This is most likely due to an error in WOExceptionPage itself. Below are the logs of first the Exception in WOExceptionPage, second the Exception in Application that triggered everything.

com.webobjects.foundation.NSForwardException [com.webobjects.jdbcadaptor.JDBCAdaptorException] dateInformation of type java.lang.String is not a valid Date type. You must use java.sql.Timestamp, java.sql.Date, or java.sql.Time: <Session> failed instantiation. Exception raised : com.webobjects.jdbcadaptor.JDBCAdaptorException: dateInformation of type java.lang.String is not a valid Date type. You must use java.sql.Timestamp, java.sql.Date, or java.sql.Time: com.webobjects.jdbcadaptor.JDBCAdaptorException: dateInformation of type java.lang.String is not a valid Date type. You must use java.sql.Timestamp, java.sql.Date, or java.sql.Time

Original Exception:
com.webobjects.jdbcadaptor.JDBCAdaptorException: dateInformation of type java.lang.String is not a valid Date type. You must use java.sql.Timestamp, java.sql.Date, or java.sql.Time

IATAC

Be careful not to catch things just to catch them

- Except for catching exceptions at the top level …
  Only catch exceptions for which you have a reasonable recovery strategy
  - Example: A FileNotFoundException might recover by allowing the user to specify a different file name
  - Example: Resource exhaustion errors such as System.OutOfMemoryException will likely need to result in process termination
- Be specific about what exceptions you are actually catching
  - Don't be tempted to catch Exceptions at the top of the Exception class
- Don't dumb down security functionality in your error handlers!

IATAC

Example from Tomcat

```
protected synchronized Random getRandom() {
  if (this.random == null) {
    try {
      Class clazz = Class.forName(randomClassic);
      this.random = (Random) clazz.newInstance();
      long seed = System.currentTimeMillis();
      char entropy[] = getEntropy().toCharArray();
      for (int i = 0; i < entropy.length; i++) {
        long update = ((byte) entropy[i]) << ((i % 8) * 8);
        seed ^= update;
      }
      this.random.setSeed(seed);
    } catch (Exception e) {
      this.random = new java.util.Random();
    }
  }
  return (this.random);
}
```

Be specific about what exceptions your code throws

- Specifying what exceptions a caller should expect from your code provides them a roadmap for what conditions they may need to handle
  - Better:
    - throws IOException, SQLException, IllegalAccessException
  - Weaker
    - throws Exception
- The weaker version makes it difficult for the caller to know what exceptions to catch

IATAC

Be careful not to quash exceptions accidentally

- C++ and Java support try/catch/finally
- Finally executes after try block regardless of whether catch was used or not
- Returning from within finally will “catch” any exceptions that have been generated within the try block
  - Unhandled exceptions will not be passed to the parent function!

```
public static void doMagic(boolean returnFromFinally)
   throws MagicException {
  try { throw new MagicException(); }
  finally {
    if (returnFromFinally) { return;}
  }
}
```

IATAC

Makes sure to log exceptions

- Exceptions often indicate unusual conditions
  - May be caused by unexpected system states, ill formed input, etc.
  - Could also be caused by intentional abuse by attacker
- Logging exceptions can greatly decrease debugging difficulty
- Logging can also be used in intrusion detection
- Rare exceptions are especially useful for debugging and ID
- General rule is to write a log entry except in the case that the exception is part of normal processing and can be completely handled within the program

```
try { doExchange(); }
catch (RareException e) {
  throw RuntimeException("This can never happen", e);
}
```

IATAC

Today's Agenda

- Error Handling, What could possibly go wrong?
- Handling return codes
- Managing exceptions
- Preventing resource leaks
- Logging and debugging
- Minor Assignment 3

IATAC

Failing to properly manage resources is a common and difficult to debug programming issue

- Programs have to manage many types of resources
  - Memory, file handles, database objects, sockets
- Improperly managing the resources can cause very difficult to debug issues
  - Tend to occur infrequently due to unusual usage scenarios or during periods of high program or system load
  - Failure often occurs in sections of the code separate from where the underlying bug exists
- Security issue is denial of service.
  - Attacker can specifically cause program to consume increasing amounts of resource until the are all used up
- Also leads to other stability / performance issues
  - E.g. There's no problem. We just reboot it once a day!

IATAC

Watch for out for returns mid-function

- Many failures to release resources are caused by functions that have multiple exit points
  - Need to ensure that every path through function results in release of all temporary resources the function has tied up
- Often easier to push resource initialization and resource release to the front and the back of the function
  - Initialization often occurs at beginning by default
  - Moving release to the end can require complex coding structures or the use of the much maligned goto

```
char* getBlock(int fd) {
  char* buf =
     (char*)malloc(BLOCK_SIZE);

  if (!buf) {
    return NULL;
  }

  if (read(fd, buf, BLOCK_SIZE) !=
      BLOCK_SIZE) {
    return NULL;
  }
  return buf;
}
```

IATAC

OO languages like C++ have much cleaner syntax for resource management

- Introduces concept of constructor and destructor
  - Constructor called when object is instantiated
  - Destructor called when object goes out of scope
- Can guarantee proper resource management
  - If all resources allocated in the constructor are released in the destructor
  - **And** no other allocation occurs in the function

What's the danger in this approach?

```
class File_handle {
    FILE* f;
  public:
    File_handle(const char* name,
      const char* mode)
      { f = fopen(name,mode);
        if (f==0)
          throw Open_error(errno);
      }
    ~File_handle() { if (f) {fclose(f);} }
    operator File*() { return f; }
    ...
};

void decodeFile(const char* fname) {
  char buf[BUF_SZ];
  File_handle f(fName, "r");

  if (!f) {
    printf("cannot open %s\n", fName);
    throw Open_error(errno);
  } else {
    while (fgets(buf, BUF_SZ, f)) {
      if (!checkChecksum(buf)) {
        throw Decode_failure();
      } else {
        decodeBlock(buf);
      }
    }
  }
}
```

With Java, there is not explicit destructor.

- Unlike C++, Java manages memory internally
  - Objects that are simply consuming memory do not need to be explicitly released as Java will release the memory automatically
  - No explicit destructor in Java
- No automatic management for other resource types (e.g. database handles)
  - Must release yourself anything you have consumed
- Finally is the most obvious method as it will always get executed at the end of a try block

IATAC

Finally can be used for resource release

- Finally is a useful constructor for resource management in Java because it will always execute at the end of the try block regardless of any errors that occur
- It use requires a few stipulations
  - Resources must be declared outside the try block (otherwise they would not be in scope inside the finally)
  - They must be forced to initialize
  - Finally must verify that the Resource has been consumed
  - Finally must be prepared to catch any exceptions the close() can throw

```
Statement stmt=null;
try {
  stmt = conn.createStatement();
  ResultSet rs =
    stmt.executeQuery(CXN_SQL);
  harvestResults(rs);
}
catch(SQLException e) {
  logger.log(Level.ERROR,
    "error executing sql query", e);
}
finally {
  if (stmt != null) {
    try {
      stmt.close();
    } catch(SQLException e) {
      log(e);
    }
  }
}
```

IATAC

Today's Agenda

- Error Handling, What could possibly go wrong?
- Handling return codes
- Managing exceptions
- Preventing resource leaks
- Logging and debugging
- Minor Assignment 3

IATAC

Logging is an important but frequently neglected aspect of application development

- When implemented well can provide key insights into operational and developmental issues with a system
  - Can provide direct insight into failures
  - Can provide evidence of attack
- When implemented poorly can assist in attacks

Login here using your username and password
(Cookies must be enabled in your browser)

Username

Password

Login

Username not found in Table Users

```
Microsoft OLE DB Provider for
ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server
Driver][SQL Server]Syntax
error converting the nvarchar
value 'login_id' to a column
of data type int.

/index.asp, line 5
```

IATAC

Use of centralized logging code makes log message creation more consistent and useful

- Centralized logging provide many benefits
  - Enables common format of log entries
  - Entries across system are logged consistently greatly adding analysis
  - Allows easy modification of logging behavior
    - Change storage type of logs
    - Add enhanced logging features (e.g. distributed logging, forward hashing, etc.)
- Many existing log frameworks exist (e.g. log4j, java.util.logging)
- If creating your own make sure to
  - Have ability to log different types of information (e.g. normal, elevated, critical). Consider making the categories editable.
  - Timestamp all entries. Makes analysis (especially across multiple systems) MUCH easier. Consider sub second accuracy.
  - Make sure to log the source of the entry (both system and application)

IATAC

Log early and often

- A logging framework adds no value if important events go unrecorded
  - Log all important activities
    - Use of multiple levels of logging would allow administrators to tune logs if volume is too large
  - An unrecorded activity will never help you find out what went wrong
- Be very careful about what you log
  - Log files may be accessible to people without any need to know information that your application processes
  - GET /cgi/usr.cgi?username=rritchey&pwd=***Secret*** HTTP/1.0" 200
  - Sensitive data such as passwords, credit cards, and SSNs should not be written to log files
- Make sure to protect your logs

IATAC

Debugging code must be removed before application is put into production

- Code is often added that has the sole purpose of allowing the developer to understand better what the code is doing
  - May allow code to be tested at the unit vs. system level
  - Could allow developer to trace execution based upon particular input
  - Many other uses
  - This is a good practice
- Must never leave this code in to production systems
  - As it was probably added without any specific design, much more likely to introduce bugs
  - May allow attacker to bypass security controls
- Segregate debugging code to enable you to remove it easily

IATAC

Today's Agenda

- Error Handling, What could possibly go wrong?
- Handling return codes
- Managing exceptions
- Preventing resource leaks
- Logging and debugging
- Minor Assignment 3

IATAC

Minor Assignment Three

- Task: Create a protected user account database.
- Detail: Applications frequently need to securely identify a user prior to allowing them to access the system. Your task in this assignment is to create a system to implement a username/ password based authentication system. The following features are mandatory :
  - Must maintain user accounts across executions (e.g. long term storage)
  - Must allow the creation, deletion of user accounts
  - Must associate a password to each user account. This should be under the control of the user
  - Must provide a demonstration application that grants user access when provide a valid username/password pair

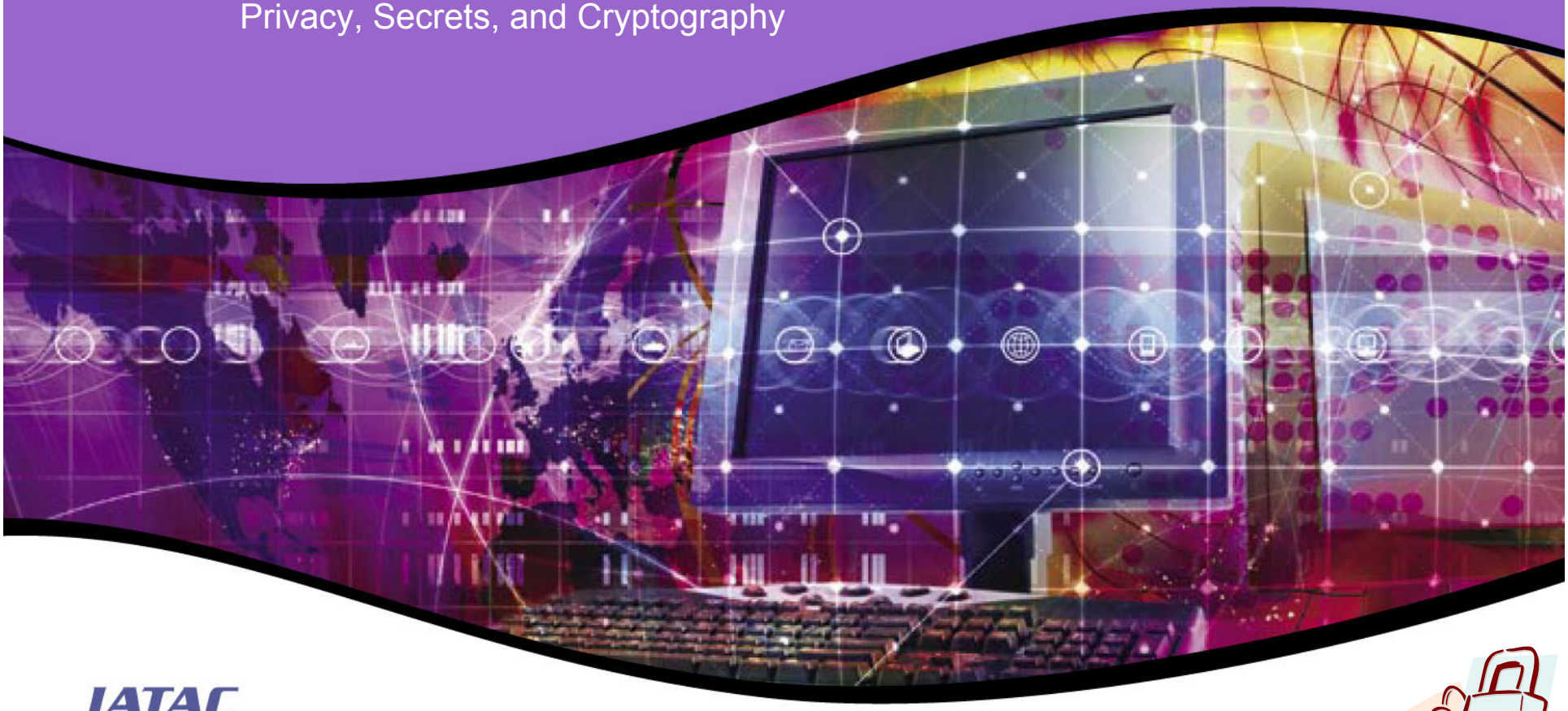
IATAC

Minor Assignment Three (cont)

- The passwords must be protected.
  - No administrator who has access to the underlying file system should be able to see clear text passwords (hint – cryptographic hashing)
  - The password file itself should be protected from tampering
- As with the previous assignments, you must reject bad input
- Permissible languages: C/C++, Java, Perl, Other with permission of instructor
- Due Date: Oct 27th

Next Thursday's Class

Privacy, Secrets, and Cryptography



IATAC

Questions?

IATAC