Taming Bugs
The Art and Science of writing secure Code

Paul Böhm

http://www.sec-consult.com/
Motivation

– Been exploiting bugs for a long time
– We keep seeing the same bugs again and again
– Better Code makes Security more interesting
– Current Software Quality sucks
  • Software Security has been neglected
Different Approaches to improving Software Security

- Education/Creating Awareness
  - “always check the buffer length”, ...

- New APIs
  - `strlcpy/strlcat`, `prepareStatement`, ...

- Abstraction
  - Automatic Memory Management, ORM, ...
Example: Buffer Overflows

• Cause
  – Program Flow ignores Memory Boundaries.
    • Out-of-Bounds Memory is written (and/or read)

• Can be triggered by
  – Array Indices (esp. in for/while loops) x[i]
  – strXcpy(), strXcat(), sXprintf, ... Style Functions
  – Pointer Arithmetics
  – ...
Education and new APIs

• The most emphasized Aspects of dealing with Buffer Overflows have been
  – new APIs (strncpy, strlcpy, strncat, strlcat, snprintf, ...)
  and
  – Education: Use strncpy/strlcpy/strncat/snprintf/...
Education/API Shortcomings

• Education and the API changes had some effects - especially since it is so easy to find strcpy() etc. bugs.

• The new APIs provide no solution to Array Indexing and Pointer Arithmetic Problems

• Education won’t help with tricky problems (e.g. well hidden off-by-one problems in pointer arithmetics) - even excellent programmers get these wrong.
Buffer Overflow Protection

- Perceived Problem:
  - The attacker is able to write past the end of the buffer:
    - Stack Canaries
  - The attacker is able to inject their own code and have it executed
    - Write XOR Execute
  - The attacker is able to execute code (own and existing) because of known addresses
    - Randomized Address Space
- These Defenses make exploitation harder but not impossible.
Defensive Programming vs. Buffer Overflows

• Real problem is that there is a possible code flow that violates Buffer Boundaries.
  – Reducing exploitability isn’t bad, but it needs to be seen as what it is: Treating the symptoms.

• To improve Security, we also need to improve the code quality.
Memory Management / Data Types

• Lots of Problems
  – noone gets memory management right, all the time!

• Can be tamed somewhat by abstracting.

• E.g. vsftpd implements its own opaque String handling.
  – Ideally no code except for the Library code itself, should be able to embarrass itself with a string buffer overflow.
Some Data Access Problems

- Abstract String Handling (store both buffer pointer, string length, and buffer length)
- Abstract Iteration (array indexing)
- Abstract Memory Allocation/Deallocation (Garbage Collection)
- Count the amount of passed Arguments for varargs. Make String formatting read-only.
- Wrap index access into buffers with bounds checking (by storing the buffer length together with the buffer in a new struct)
- Use highlevel Integer types to avoid Integer Overflows and signedness misinterpretation. (e.g. transparently switch the data representation when a string grows too large)
Bug Economies of Scale

- You don't strcpy() once, you don’t free() once, you don’t do pointer arithmetics once.
- Bugs that fall within well known Bug Classes pop up all over the place.
  - The more code you write, the more opportunities to fsck up you have.
  - Eventually even good programmers make mistakes.
- We need approaches that allow us to write as little bug prone code as possible.
The Nature of the Beast: Bugs

• Given the same task and the same tools many programmers will
  • choose similar implementation strategies
  • make similar mistakes

• For most Bug Classes is true:
  – You’ve got to be careful of similar mistakes at lots of places
    • The amount of critical code portions scales with the amount of code.

• Attackers and Pen-Testers look for those common mistakes.
Standards using ASN.1

*) SNMP – Simple Network Management Protokol
*) VOIP/H323
*) SSL/TLS – Secure Socket Layer / Transport Layer Security / HTTPS
*) NTLM – NT Lan Manager Authentication Service
*) ASN.1 Compiler
*) S/MIME – Secure/Multipurpose Internet Mail Extensions
*) IKE – Internet Key Exchange (VPN)
*) Kerberos Authentication Service
*) LDAP – Lightweight Directory Access Protocol
*) CIFS/SMB – Common Internet File System / Samba
Security Vulnerabilities in ASN.1 Implementations

*) SNMP – Simple Network Management Protokol
   + CA-2002-03 (ADTran, AdventNet, ADVA, Alcatel, Allied Telesyn, APC, Aprisma, Avaya, BinTec, BMC, CacheFlow, 3Com, ucd-snmp, Cisco, CNT, Compaq, Computer Associates, COMTEK, Concord, Controlware, Dart Communications, Microsoft, Lotus Domino, ...
   + CAN-2004-0918 (Squid Web Proxy SNMP ASN1 Handling)

*) VOIP/H323
   + DoS in Vocaltec VoIP gateway in ASN.1/H.323/H.225 stack

*) SSL/TLS – Secure Socket Layer / Transport Layer Security / HTTPS
   + Microsoft ASN.1 Library Bit String Heap Corruption
   + Microsoft ASN.1 Library Length Overflow Heap Corruption
   + CAN-2003-0543 - Integer overflow in OpenSSL 0.9.6 and 0.9.7 with certain ASN.1 tag values.
   + CAN-2004-0401 - libtASN1 DER parsing issue (GNUTLS)

*) NTLM – NT Lan Manager Authentication Service
   + CAN-2003-0818 - Multiple integer overflows in Microsoft ASN.1 library (MSASN1.DLL)
Security Vulnerabilities in Standards that use ASN.1 (Continued)

*) ASN.1 Compiler
   + BID-11370: ASN.1 Compiler Multiple Unspecified Vulnerabilities

*) S/MIME – Secure/Multipurpose Internet Mail Extensions
   + CAN-2003-0564: Multiple vulnerabilities in multiple vendor implementations [...] and possibly execute arbitrary code via an S/MIME email message containing certain unexpected ASN.1 constructs

*) IKE – Internet Key Exchange (VPN)
   + BID-10820: Check Point VPN-1 ASN.1 Buffer Overflow Vulnerability

*) Kerberos Authentication Service
   + CAN-2004-0644: The asn1buf_skiptail function in the ASN.1 decoder library for MIT Kerberos 5 (krb5) 1.2.2 through 1.3.4 allows remote attackers to cause a denial of service

*) LDAP – Lightweight Directory Access Protocol
   + CA-2001-18 (iPlanet, IBM, Lotus Domino, Eudora WorldMail, MS Exchange, NA PGP Keyserver, Oracle Internet Directory, OpenLDAP, ...)

*) CIFS/SMB – Common Internet File System / Samba
   + CAN-2004-0807: Samba 3.0.6 and earlier allows remote attackers to cause a denial of service via certain malformed ASN.1 requests
Dealing with Bugs

• Don't deal with bugs. Deal with Bug Classes instead.

• If you find a bug
  – Fix it
  – Then think about how you can make sure you'll never have another bug like that in your code.
  -> put yourself on rails!
Abstraction is the Key

• Solution Case Study: vsftpd

• (mostly) Opaque String Handling

```c
struct mystr
{
    char* PRIVATE_HANDS_OFF_p_buf;
    /* Internally_EXCLUDES_trailing null */
    unsigned int PRIVATE_HANDS_OFF_len;
    unsigned int PRIVATE_HANDS_OFF_alloc_bytes;
};
```

• Lots of special case routines
  – `str_netfd_read()`
  – `str_chmod()`
  – `str_lstat()`
  – `str_lstat()`
  – `str_syslog()`
  – `str_open()`
  – ...

Generalizing Abstraction

• vsftpd style abstractions haven’t caught on much in the C World
  – Too much special case code to be universally usable.

• Many Higherlevel Languages provide a more general Approach to tackling the problems of memory access and management.
Bug Classes dealt with by abstracting MemoryMgmt/Data Types

- Stack Overflows
- Heap Overflows
- Off-by-one
- Double free()
- Missing Memory initialization
- Format Strings
- Unchecked indices, array access
- Integer Overflows
Source: “Software Security is Software Reliability”, Felix Lindner, CACM 49/6
Using Abstractions for Defensive Programming

• Mistakes become less likely.
  – Fewer places where you can make mistakes.

• You can still shoot yourself in the foot if you want to.
  – But you've got to try harder!

• If you abstract what you are trying to do, code auditing becomes easier.
  – Even program-driven static analysis works best if there's little guesswork involved.
Performance Downsides of Abstraction?

- Fortran Vectors vs. GPU
- 150 parallel Instructions on the P4
  - manual optimization?
- Wrong Java Abstraction (highlevel semantics on lowlevel datatype)
- IronPython .net Implementation faster than the CPython Implementation. Same goes for Pypy
- More Data on what you want to do helps the compiler optimize!
  - > Abstraction is good!
How to squash Bug Classes

- Use Abstractions that make it easy to “do the right thing”™
- Define that use of bug-prone APIs and syntax are bugs.
- Use APIs that are easy to audit and if possible supportive of static analysis.
- Use Code Audits and Static Analysis for Regression Testing.
How to deal with other Bug Classes

• SQL/XPATH/LDAP Injection
• Insufficient Hamming-Distance
• Programming Language Magic
• Insufficient Expressiveness
• Cross Site Request Forgeries
• Cross Site Scripting
• Path Traversal
• …
Insufficient Expressiveness

• Negative Example: Programmer wants to iterate over the Elements of a list.
  – for \((x = 0; x <= \text{argc}; x++)\)
    doSmtn(argv[1]);
  – > instant Off-by-One + another bug
  – instead of
  – for (elem in argv):
    doSmtn(elem)

• -> A highlevel construct, iterators, abstract the problem.
Insufficient Expressiveness

• Negative Example:
  – Programmer wants to list all Files in a Directory.
• `while (false !== ($file = readdir($handle)))`
  echo "$file\n";
  instead of
• `for x in os.listdir("."):`
  print x
Hamming-Distance

• if (x == 5) { /* ... */ } is too close to
• if (x = 5) { /* ... */ }

• char *x[ ] = {"as", "fg", "xc", "b"}; too close to
• char *x[ ] = {"as", "fg", "xc" "b"};
Programming Language Magic

• Negative Examples:

• Userinput gets automatically stored in global Variables:

• http://xxx/foo.php?blah=foo
  -> implicit $blah = "foo";
Programming Language Magic

- fopen(), include(), understand URLs.

  - include($subsite) executes php code which gets downloaded from a remote server.

- If you disable this feature, you're on your own if you want to download something via HTTP.
Programming Language Magic

• Undefined Variables get automagically defined as empty on use.

• When two Variables of differing type get compared one of them gets implicitly converted:
  • e.g. $id == “my_string” is true if
    • $id is a string that contains "my_string" or
    • If $id is an integer with value 0, "my_string" gets converted to an int of value 0.
Injection Problems

- SQL/LDAP/XPath/… Injection,
- XSS

- Are all caused by injecting Data of one Type (often plaintext), into Data of another type (SQL, HTML, …) – without conversion
String Types

• What is a String ‘Type’?
  – Strings are just strings, right?

• Strings are just random bytes strung together
  – However they acquire meaning by the way they are used

• For SQL/HTML/… we already know how we’re gonna use them.
String Types

- Injection Problems are caused by forgetting to convert Data for its dedicated use.
  - We have to always escape(uservar) for HTML, or escapeQuotes(uservar) for SQL.
    - If we forget just once, we have a problem.
- If we’re already talking about String Types – why not just use the type system to remind us to convert?
  - HTMLString, SQLString, …
Cross Site Scripting

• Data that comes from users is of type ‘str’
  – That’s just a string without semantic meaning

• All strs get auto-converted to HTMLString before being output.

• All Strings stored in the database are of type ‘str’, unless specified otherwise in the Database Model.
  – Alternatively we can just unescape in the Templating Language
Cross Site Scripting

• XSS Blog Demo
• XSS Protection Demo
• (Static Analysis)
SQL Injection

- PHP

```php
$sql = "SELECT * FROM customers WHERE name = "" . $_POST['name'] . ""; 
$query = mysql_query($sql) or die("Database error!");
```
SQL Injection

- Java
  Statement stmt = con.createStatement();
- String sql = new String("SELECT * FROM customers WHERE name = " + request.getParameter("name") + "")
- ResultSet rset = stmt.executeQuery(sql);
SQL Injection – PHP fixed

- $sql = "SELECT * FROM customers WHERE name = '' . mysql_real_escape_string($_POST['name']) . '"";
- $query = mysql_query($sql) or die("Database error!");
Better abstraction than in PHP:

```java
PreparedStatement pstmt = con.prepareStatement("SELECT * FROM customers WHERE name = ?");
pstmt.setString(1, request.getParameter("name"));
ResultSet rset = pstmt.executeQuery();
```
SQL Injection – Abstracting further

- DAO – Data Access Objects
  - Decouple Data Access logic from Business Logic
  - Slightly better to maintain, because SQL is only used in a limited area of your code
  - Still as easy to make SQL Injection Bugs
  - Lots of glue code!
SQL Injection – Going further

- ORM Object Relational Mappers
  - Hide the SQL from Programmers (for most cases)
  - Where you don't write SQL, you can't create SQL Injection problems
  - Queries look like this:

```python
Customer.objects.get(name=name, birth_date__year=1980).order_by('-birth_date', 'name')
```
SQL Injection – Demo Time

- Demo
SQL Injection – Regression

• Both prepared statements and ORM make statical Analysis for Regression Testing easier

• For prepared statements, check if the template is a constant.

• Doesn’t work with generated SQL -> use as little as necessary.
Path Normalization

• The Problem:
  – userSuppliedFilename = "/../../etc/passwd";
  – open("/var/www/data/"+userSuppliedFilename);
• The Solution:
  – Path Normalization:
    • normalize("foo/1/2/3/4/../7") -> "foo/1/2/7"
    • absolute("data/file.txt") -> 
      "/var/www/data/file.txt"
    • normalize(absolute(userPath)).startswith("/valid/directory/root") ?
Path Normalization

Diagram showing various paths being normalized to a common format. The paths include:
- a/..b/x.png
- /etc/passwd
- illustr.jpg
- a/b/c/d/..../e/f
- a/b.png
- x4223.html
- ../../etc/passwd
- /var/www/data/frob.txt

Normalized Paths:
Path Normalization

• Buggy Demo

• Fix Demo

• Further Abstraction
  – openWithinPath("/var/www/data", userDir)
  – Lends itself well to auditing.
Cross Site Request Forgeries

- Example (GET): http://web.example.net/changePass?new Pass=<smtn>
- POST most often realized with javascript in IFRAME.
- CSRF Demo
- CSRF Middleware ProtectionDemo
There is more

• Layered Design
  – Split up code to run with least privilege
  – Protocol Parsing is bug prone - don’t let it run with full privileges

• Write highlevel code that is easy to audit, and abstractions that clearly say what you want to do.
  – The more info goes into the code, the easier auditing both by people and programs gets.

• But get the basics right first: Don’t repeat yourself in bug-prone code-parts.
Questions?