Computer Security - Tutorial 3: Exploitation

Myrto Arapinis and Joseph Hallett

18th March 2015

Memory Safety

```
/* Copy every step'th int from src to dst */
void vuln(int *dst, int *src, int n, int step)
{ int i;
  for (i = 0; i < n; i += step)
    dst[i] = src[i];
}
```

Question 1 Depict the stack frame of a call to `vuln()` before the execution of the `for` loop. You can assume the code is being executed on a 64-bit Linux machine (like those in the lab), with the size of the `int` datatype being 8 bytes. Standard calling conventions are being used.

Question 2 Assume `src`, `n` and `step` are under control of the attacker, but that there are additional checks that `src` and `dst` are non-NULL pointers, and that `n` $\leq$ `sizeof(*src)` and `n` $\leq$ `sizeof(*dst)`. The code still has a memory safety vulnerability. Describe it.

Question 3 How could an attacker exploit this vulnerability (*i.e.* what parameters could they give to `vuln()` to mount an attack, assuming they have control of `src`, `n` and `step` as before). How much damage can they do with each invocation of the function? Where might they want to do this damage to? What additional restrictions might they have for your attack to work?

Question 4 Suppose `vuln()` writes only a `char` (1 byte) each time. Is your attack still possible? What extra conditions would you need for your attack to work?

Question 5 Modern compilers insert stack canaries to prevent buffer overflow attacks. Would they prevent your attack? Why?

Script Injection

Alice has written a program in Java to create a forum for herself and her friends. Unfortunately the code is vulnerable to several attacks.

The program consists of a website, some Java based logic, and a database containing her user's login information and the forum posts themselves. When users attempt to login their username and password is sent to the following function:

```
public void login(final String username, final String password) {
    final Connection connection = getConnection();
    final String hPassword = hashPassword(password);
    String sqlString = "SELECT *"                          +
                       "  FROM db_users"                   +
                       " WHERE username='"+username+"'"    +
                       "   AND password='"+hPassword+"'" ;
    Statement statement = connection.createStatement();
    ResultSet rs = statement.executeQuery(sqlString);
    if (!rs.next()) { throw new SecurityException(); }
    // Login as username
}
```

Question 1 The code is vulnerable to an SQL injection attack. Describe how you could attack the code to enable you to login as any user without knowing their password.

Question 2 How should the code have been written? What defensive programming techniques would prevent this attack?

Alice stores forum posts inside her database. Before inserting them into the database she takes care to remove all `<script>` and other tags to prevent Javascript being injected. She does, however, allow `<b>` tags so her users can use a little bit of formatting in their posts. To do this she compiled a list of all other HTML tags and created a series of regular expressions of the form:

```
s!</?script(\s\w+(\=\".*\")?)*\>!!gi
```

Question 3 Describe how you could attack the forum using the `<b>` tag to allow inject arbitrary Javascript onto a page. What conditions would have to be met for your code to run?

Question 4 Suppose the `<b>` tag is also removed. Is it still possible to inject Javascript into the page?

Question 5 The regular expression for this question came from an accepted *Stack Overflow* answer about how to remove tags. How should the HTML stripping code have been written?

Cookie Stealing

Alice's forum has more problems. Once her users have logged in she gives them a cookie which authenticates them for their session. If a user presents this cookie to her server Alice knows they're logged in. If a user presents a different user's cookie then Alice rejects their login.

Question 1 How could Mallory use a script injection attack from earlier to steal a cookie from a user.

After a while Mallory has collected a few cookies! She observes the following:

```
User:   'bob'
Cookie: '9f9d51bc70ef21ca5c14f307980a29d8.6'

User:   'claire'
Cookie: '182e500f562c7b95a2ae0b4dd9f47bb2.12'

User:   'david'
Cookie: '182e500f562c7b95a2ae0b4dd9f47bb2.13'

User:   'bob'
Cookie: '9f9d51bc70ef21ca5c14f307980a29d8.30'
```

If Mallory attempts to authenticate with Bob's first cookie her login fails, but if she uses his second cookie it still works.

Unfortunately Alice hasn't fallen for her trap and Mallory wants to steal her admin account.

Question 2 Suggest how Alice makes her cookies. Can you guess what a valid cookie for Alice might be?

Question 3 How should Alice be generating her cookies?

Alice goes away and fixes the injection attacks, and makes her cookies by generating a random string every time a user logs on. She makes sure she only serves her website over HTTPS using a self signed certificate.

Question 4 Can Mallory still steal cookies? (Hint: perhaps she could team up with *Eve*?) What extra restrictions does she have?

Question 5 How much better are Alice's new cookies?