Software Vulnerabilities

Lecture Notes

by Matthias Vallentin

September 4, 2008

1 Access Control Matrix (ACL)

	pid 123	pid 124	pid 125	printer	/tmp/foo	/tmp/bob
pid 123						r
pid 124					r	r
pid 125					r,w	r,w,*own*

Table 1: Samle ACL

- **Rows**: subjects (active entities, take actions)
- **Columns**: subjects & objects (passive entities, can be acted upon)
- In most cases, ACL is enforced by the OS
- Most systems do not store the entire matrix, because of
  1. its vast size
     ⇒ sparse matrix storage used instead (e.g. columns stored in list)
  2. the profusion of subjects
     ⇒ optimization: each process has a PID and a UID. Use the UID rather than the PID to create equivalence classes of subjects with the same UID.
     Example: `pid 123` and `pid 123` were started by Alice (and thus have Alice's UID) and `pid 125` was started by Bob. Instead of using the three PIDs, merely the UIDs of Alice and Bob are stored. ⇒ another optimization: use *roles/groups* to reduce the number of subjects.

- Dealing with permission changes:
  - Example: Bob creates a new file `/tmp/bob` and has the *ownership* right for the file, allowing him to grant specific rights to other users (e.g. allow Alice to read the file).
- Which objects can Bob access in the future? This question is undecidable, as it requires to examine all applications that could potentially grant a certain access right to Bob. Hence, practical analyses include only a restricted set of verifiable code pieces.

2 Software Vulnerabilities

2.1 Buffer Overflows

- Accounted for largest share of software vulnerabilities in the past
- The various exploitation techniques evolved over time, it begun with the famous paper "Smashing the Stack for fun and profit" by Aleph One.
- 2006: buffer overflows no longer number 1 vulnerability → down to 4 (8%)
  ⇒ web-based exploits are now the low-hanging fruit.
- "If you have a buffer overflow in your code, assume it's exploitable."

2.1.1 Heap Overflows

- Basic idea: instead of corrupting management information on the stack, manipulate chunk meta data on the heap.
- Refer to the slides in [1] for visualization.
- **Heap Spraying**: `http://en.wikipedia.org/wiki/Heap_spraying`

2.2 Format String Vulnerabilities

```
f()
{
    char buf[512];
    gets(buf);
    printf(buf);  /* instead of printf("%s", buf) */
}
```

- No type-safe argument passing conventions in C.

- *varargs* mechanism allows functions to accept any number of arguments (e.g. printf) by "popping" as many arguments off the call stack as they wish
- Problem: it is assumed that the early arguments indicate how many additional arguments are to be popped, and of what types.
- Exploitation: Supply format variables in input buffer, `printf` will look for arguments that are not there
  - Example buffer: `"%d %d %d %d %d"` looks for arguments 1, 2, 3, 4, 5 which are not present in the above code fragment. This basically enumerates the stack, no harm yet
  - `%n`: *write* the number of characters read so far, expects a `int*`.
  - Goal: craft a format string that overwrites a specific location in memory via `%n`.
  - Refer to the slides in [2] for visualization.

2.3 Integer Overflows

```
unsigned int n = get_word();
int *a = malloc(n * 4);
int i;
for (i = 0; i < n; i++)
    a[i] = get_word();
```

- Example code parses a typical length-value protocol format. The first dword (here: `n`) represents the size of the following data.
- Choosing $2^{30} + 1$ for `n` causes an exact overflow
  $\Rightarrow$ results in a heap overflow in this case.

3 Mitigation Strategies

- Do not use C, but
  - legacy code: high rewrite costs, 10-30% of bug fixes introduce new bugs
  - performance
  - training: programmers make more mistakes in languages they do not know well.
  - integration: many applications rely on applications written in C.

- Input sanitization: never trust user input
- Educate the programmers: avoiding mistakes in advance
- Target mitigations (tbd)
- Never mix control and data channels ⇒ command injection attack
  - Example 1, Pseudo-Perl: `system("mail" + $email_addr)`
    ⇒ your email address: `; /bin/rm -rf /`
  - Example 2, SQL Injection: `SELECT * FROM users WHERE username = $user AND password = $pass`
    ⇒ your password: `foo OR 1=1`
  - Example 3, Captain Crunch: AT&T payphones used inbound signaling at 2600Hz after a coin has been inserted. A whistle in the "Captain Crunch" cereal blew at the same frequency, enabling free phone calls.
    ⇒ fix: separation of control and data channel

References

[1] Matthias Vallentin. On the Evolution of Buffer Overflows. Munich, May 2007.

- Paper: `http://matthias.vallentin.cc/cw/buffer_overflows.pdf`
- Slides: `http://matthias.vallentin.cc/cw/buffer_overflows_talk.pdf`

[2] scrut / team teso. Exploiting Format String Vulnerabilities. September 2001. `http://julianor.tripod.com/bc/formatstring-1.2.pdf`