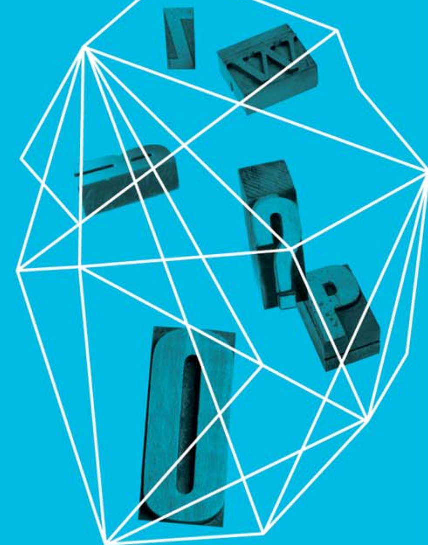
RSA CONFERENCE ASIA PACIFIC 2013

Security in knowledge



SECURING SOFTWARE AGAINST LIBRARY ATTACKS

Roland Yap

School of Computing

National University of Singapore

ryap@comp.nus.edu.sg

Session ID: DAS-W05

Session Classification: Advanced

Untrusted Libraries

- Software developer Bob wants to write a photo gallery
  - Bob finds a library for PNGs
  - Bob might not trust the library
    - can it steal photos? tamper with photos?
- What can Bob do?
  - Analyse the source of the library?
    - can vulnerabilities/malicious behavior be found?
  - What if no source?
- In practice – just use the library

Image library

Massive Use of External Software Libraries

- GoogleChrome uses 115 **external** libraries; Firefox uses 171
- Software Plug-ins:
  - A framework to allow third party modification
  - E.g. Adobe Photoshop, Winamp, GStreamer, GIMP, Kernel Driver
- Browser Extensions: Flash, Java, QuickTime, Real Player, …

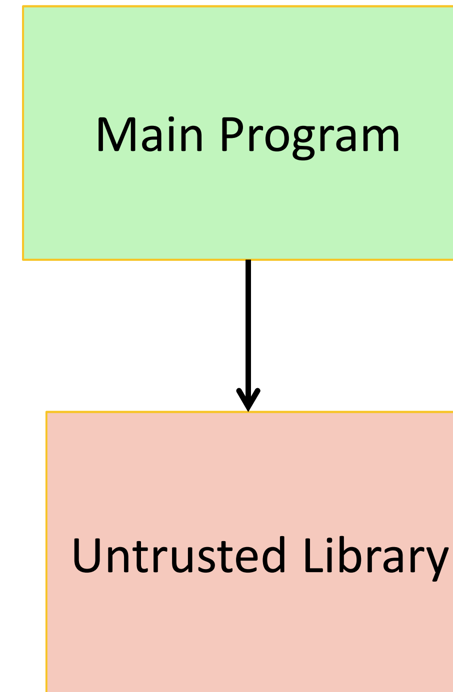
- From www.**libpng**.org: 103 Web browsers, 154 image viewer and 21 hardware use libpng. They gave up counting them 6 years ago
- From www.**gzip**.org: *"This list is getting pretty long. Eventually, it may be easier to list the applications that don't use zlib!"*

Reported Vulnerabilities in Libraries

- libpng vulnerabilities in 2011
  - See: **`http://www.libpng.org/pub/png/libpng.html`**
  - Denial of Service: CVE-2011-3328, CVE-2011-3045, CVE-2011-2692, CVE-2011-2691, CVE-2011-2501
  - Denial of Service: CVE-2011-3328, CVE-2011-3045, CVE-2011-2692, CVE-2011-2691, CVE-2011-2501
  - Other years: 2010, 2009, 2008, 2007, …
- Malicious Plug-ins
  - **Trojan.PWS.ChromeInject.A**: Firefox plugin that collects a user's passwords from banking sites
  - **Heuristic.BehavesLike.Exploit.CodeExec.I**: worm disguised as VLC plugin libwav_plugin.dll

Some Definitions

- Main Program
  - trusted code
  - full privileges
- Untrusted Code
  - library code, plugin, ...
  - reduced privileges



Software Fault Isolation (SFI)

- SFI – prevent library from modifying memory outside its own space
  - library prevented from writing to memory contents of **`main`**
  - sandbox library to its own memory space

- **What about system calls**?
- **What about tight interactions**?
  - Passing parameters by reference + return by reference
  - Callbacks – library calls function in **`main`**
  - Long jump + exceptions
  - Shared Global variables

Example of Tight Interactions

- an example using **`libpng`**
- shows various tight interactions between **`main`** and **`libpng`**

```c
static void row_callback(png_struct *png, png_bytep new_row,
   png_uint_32 row_num, int pass) {  // display the row }
int main (void) {
  FILE *fp = fopen("foo.png", "rb");
  png_struct *png = png_create_read_struct(...);
  png_info *info = png_create_info_struct(png);
  if (setjmp(png_jmpbuf(png))) {
    png_destroy_read_struct(&png, &info, NULL);
    close(fp); return 1;
  }
  png_set_progressive_read_fn(ptr, ..., row_callback, ...);
  while (1) {
    char buff[1024];
    size_t len = fread(buff, 1, 1024, fp);
    if (!len) break;
    png_process_data(png, info, buff, len);
  }
  png_destroy_read_struct(&png, &info, NULL);
  fclose(fp); return 0;
}
```

libpng **API**

```
static void row_callback(png_struct *png, png_bytep new_row,
   png_uint_32 row_num, int pass) {  // display the row }
int main (void) {
  FILE *fp = fopen("foo.png", "rb");
  png_struct *png = png_create_read_struct(...);
  png_info *info = png_create_info_struct(png);
  if (setjmp(png_jmpbuf(png))) {
    png_destroy_read_struct(&png, &info, NULL);
    close(fp); return 1;
  }
  png_set_progressive_read_fn(png, ..., row_callback, ...);
  while (1) {
    char buff[1024];
    size_t len = fread(buff, 1, 1024, fp);
    if (!len) break;
    png_process_data(png, info, buff, len);
  }
  png_destroy_read_struct(&png, &info, NULL);
  fclose(fp); return 0;
}
```

Returning Result by Reference

Passing Parameter by Reference

```c
static void row_callback(png_struct *png, png_bytep new_row,
   png_uint_32 row_num, int pass) {  // display the row }
int main (void) {
  FILE *fp = fopen("foo.png", "rb");
  png_struct *png = png_create_read_struct(...);
  png_info *info = png_create_info_struct(png);
  if (setjmp(png_jmpbuf(png))) {
    png_destroy_read_struct(&png, &info, NULL);
    close(fp); return 1;
  }
  png_set_progressive_read_fn(ptr, ..., row_callback, ...);
  while (1) {
    char buff[1024];
    size_t len = fread(buff, 1, 1024, fp);
    if (!len) break;
    png_process_data(png, info, buff, len);
  }
  png_destroy_read_struct(&png, &info, NULL);
  fclose(fp); return 0;
}
```

Long Jump

Callback

Library Sandboxing Solutions

- Google Native Client (NaCl)
  - Designed to sandbox untrusted modules in **browser** (Chrome)
  - **SFI-based**
  - can be used for sandboxing libraries (but ...)
  - Well supported by Google, may not be so easy to use
  - Comes with tool chain and tool support
- CodeJail
  - Research prototype
  - Developed at NUS
  - New memory model, differences from SFI
  - **Supports Tight Interactions**

Native Client Basics

- recompile module with NaCl tool chain
  - generates **safe** machine code
  - does not support all programs
    - only safe subset of machine instructions
    - disallowed instructions: syscall, int, lock, ...
- Code in NaCl sandbox
  - can only access NaCl created region of memory
  - memory access errors cause exceptions
    - hardware exception handling limitations
  - no system calls allowed
    - NaCl supports restricted `libc` with system calls run outside sandbox
  - NaCl compiled code reasonably efficient – rewrites potentially unsafe memory + jmp instructions to safe sequences

CodeJail Basics 1

- Novel Memory Model
  - different from SFI
  - `main` + library share same **address space**
    - contents of memory differ
    - `main` can read/write untrusted library memory
    - untrusted library cannot write to memory of `main` – leads to separate copy
    - library can **read** memory of `main`
    - library memory is persistent – supports library global variables
  - designed to **support tight interactions**
    - except for library writing to `main`'s memory
    - shared global variables used in controlled way supported by APIs
- Implemented with operating system memory protections

CodeJail Basics 2

- Can work with any reasonable library
- No recompilation
- CodeJail API is used to interact with library
  - but can be made **transparent** with rewritten library wrappers
- System calls restricted using a system call policy
  - library can run with reduced privileges

CodeJail Implementation

- Linux prototype
- protection guarantees due to Unix/Linux kernel mechanisms + virtual memory protection
- reasonable overheads
  - overheads commensurate with % calls and tight interactions
- transparently run real programs + real libraries with tight interactions
  - tested libraries with tight interactions:
    - `libpng`, `libtiff`, `libbzip2`, `libexpat`
  - `Firefox` with `libpng` sandboxed

Native Client vs CodeJail

NaCl

- source needed
- recompile with NaCl toolchain
- source modification needed
  - library need to use NaCl mechanisms
  - `main` program may need changes
- compatibility
  - SFI model, tight interactions not allowed
- implementation
  - architecture specific, requires de-optimizations
  - efficient - only a few percent overhead on SPEC benchmarks

CodeJail

- binaries sufficient
- no recompilation – existing binaries
- no modification, transparent to main
  - API wrapper library may be needed
  - can also write programs with CodeJail API
- compatibility
  - supports many tight interactions
- implementation
  - OS based, portability based on OS mechanisms, overhead ~ page fault + etc
  - not as efficient as NaCl to transfer context from `main` to library

Security Guarantees

- Both SFI solutions (Native Client) and CodeJail
- memory in `main` cannot be modified by library
  - ensures integrity of `main`
- system privileges are restricted in library

Attack on Library

- Suppose library is malicious or has exploited vulnerability
  - arbitrary code execution in library
  - normally bad news
- Sandboxed library/plugin
  - arbitrary code execution in library
    - more restricted in NaCl
  - cannot modify data in `main`
    - cannot write to `main` stack + heap + globals
  - cannot change execution in `main`
  - only has privileges of library
    - cannot escalate privileges

Library Security Checklist

- Does your code use libraries?
  - Do you use plugins?
  - Do you use loadable runtime modules?
  - Can external libraries be loaded?
- Are the libraries trusted?
  - should they be trusted?
  - are exploitable vulnerabilities possible?
- Can your libraries be modified or substituted?
  - library path attacks
- Do you have source code for libraries?
- Do you have source code for your programs/applications?

How to protect yourself

- SFI solutions work
  - Native Client – particularly if its a browser plugin
  - may require source code + rewriting of main + library
    - may be hard if there is significant tight interactions
  - runtime overheads low modulo code changes
    - may be higher if code is large
    - significant data copying/transfers needed
- Libraries with tight interactions
  - CodeJail-like solutions
  - not all tight interactions can be supported
    - reasonable programs + libraries may be transparent to sandbox
  - CodeJail still alpha stage

Questions?