International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 05 Issue: 08 |August 2018 www.irjet.net p-ISSN: 2395-0072

Social Engineering Using Restoration as a Threat to Commit Cyber Crime

Dr. ANISHA KUMAR

[1]*PRINCIPAL & Professor, ST. Joseph's College For Women , Mysore-570 029(India) .*

Abstract - *Social engineering is a very affective area of computer crime which is unnoticed in most of the organizations and in few places it becomes too late to accept the fact that a person is being robbed of his personal and important information. There are different levels of methods to activate social engineering such as psychological and physical attacks. Despite the increase in government compliance requirements and the proliferation of security tools, companies continue to underestimate the threat of losing data through social engineering. This paper contributes an understanding of the different ways how social engineering is activated and the importance of the concept of restoration which leads to social engineering. A few aspects of tracing the deleted information with restoration software is also mentioned. This method is also called trashing.*

***Key Words*: Social engineering, hacking, restoration, internet**

1. INTRODUCTION

Social engineering is essentially the art of gaining access to buildings, systems or data by exploiting human psychology, rather than by breaking in or using technical hacking techniques. For example, instead of trying to find a software vulnerability, a social engineer might call an employee and pose as an IT support person, trying to trick the employee into divulging his password. Security is all about trust and Social engineering is a systematic process that is planned to break this trust. In the present scenario of cyber crime social engineering is a process that has been the major source to make the hackers use psychological tricks and gain important information which hold high value.

1.1 SOCIAL ENGINEERING ATTACKS

Social engineering attacks occurs in two different levels such as psychological and physical attacks. In the psychological level the hacker observers the employees moves and also tries to see the password typed by the employee. In the physical level of attack the hacker shows his physical presence in disguise and tries to be a person in position, wherein the intruder struts through the office until he or she finds a few passwords lying around and emerges from the building with ample information to exploit the network from home later that night.

1.2 CATEGORIES OF SOCIAL ENGINEERING

The concept of social engineering can be executed in different categories such as through phone, online, trashing, persuasion and reverse social engineering. Social engineering through telephone is the most common type where a hacker will call up and imitate as a person in position by which the hacker will collect all relevant details from the other end converser. It is also possible that the hacker will try as helping any transaction details over the phone by answering the queries and finally will reach a conversation of forcing the converser to give his or her personal details such as user id and password to open the lock or track the problem.

2. ONLINE SOCIAL ENGINEERING

Online social engineering is activated by using the internet which is a fertile ground for the culprits to harvest passwords. When the users repeat the use of single password on every account it becomes easy to hack the account to accesses important information. Persuasion is a method of creating a perfect psychological environment for attack. This activity includes a number of methods to gather information such as impersonation, Ingratiation, conformity, diffusion of responsibility and friendliness. Whatever methods are used social engineers main idea behind is to make the person disclose the important and sensitive information. Advanced method of social engineering is termed as Reverse Social Engineering where hacker creates a persona that appears to be in position of authority and makes it a planned way of gathering information from the employees.

2.1THRASHING AND RESTORATION

Thrashing also called as dumpster diving is a very threat able method which uses the concept of restoration to retrieve useful information from hard drives which were outdated. Information are collected as a leak in the trash such as companies private phone books, policy manuals, organization calendars of meetings, printouts of source code, login names and mainly from outdated hardware. Based on the survey done on few companies in Mysore city and on individual employees it has been proved that most of the information are lost through the method of trashing.

The method of restoration or thrashing which can retrieve information possibly leads to cyber crime. The problem of

thrashing can be overcome by the process of defragmentation and aging. As an example the process of aging can be well explained in an Ethernet device which is also applicable for any information stored in the cache entries. Every Ethernet devices has a unique byte number called the medium access control (MAC) address, assigned to it for addressing. The devices on a LAN communicate with each other only with this number. If a system needs to send data to another system, the kernel generate an address resolution protocol (ARP) packet containing the IP address of the destination system. This packet is broadcast to all other systems on the Ethernet network. A broadcast uses a special network address to signal that all host should receive and process the packet. Only the system whose IP address matches the IP address of ARP request responds and sends back its MAC address to the system that did the query. For efficiency the host caches the IP MAC address pair in an internal table. The cache entries are aged, so that they eventually get removed from the cache if an access to that system is not required in the given time. In this way hosts that are removed from a network are eventually forgotten. This describes that the process of aging can avoid thrashing to some extend.

Details of the c source code created for a recovery software is given below. This is a work done to retrieve information and defragmentation to be applied to overcome thrashing.

2.2. HEADER FILES FOR RESTORATION

The list of header files included in the source programme is given below :

#include <ctype.h> **// character type for upper and lowercase etc.**

18 #include <errno.h>

19 #include <fcntl.h>**// file control option header file for the purpose of setting flags etc.**

20 #include <getopt.h>

21 #include <limits.h>

22 #include <linux/input.h>

23 #include <stdio.h>

24 #include <stdlib.h>

25 #include <sys/reboot.h>

26 #include <sys/types.h>

27 #include <time.h>

28 #include <unistd.h>

29

30 #include "bootloader.h"

31 #include "commands.h"

32 #include "common.h"

33 #include "cutils/properties.h"

34 #include "firmware.h"

35 #include "install.h"

36 #include "minui/minui.h"

37 #include "minzip/DirUtil.h"

38 #include "roots.h"

39

40 static const struct option OPTIONS[] = {

41 { "send_intent", required_argument, NULL, 's' },

42 { "update_package", required_argument, NULL, 'u' },

43 { "wipe_data", no_argument, NULL, 'w' },

44 { "wipe_cache", no_argument, NULL, 'c' },

45 };

46

47 static const char *COMMAND_FILE = "CACHE:recovery/command";

48 static const char *INTENT_FILE = "CACHE:recovery/intent";

49 static const char *LOG_FILE = "CACHE:recovery/log";

50 static const char *SDCARD_PACKAGE_FILE = "SDCARD:update.zip";

51 static const char *TEMPORARY_LOG_FILE = "/tmp/recovery.log";

52

53 /*

54 * The recovery tool communicates with the main system through /cache files.

55 * /cache/recovery/command - INPUT - command line for tool, one arg per line

56 * /cache/recovery/log - OUTPUT - combined log file from recovery run(s)

```
57 * /cache/recovery/intent - OUTPUT - intent that was passed in

58 *

59 * The arguments which may be supplied in the recovery.command file:

60 * --send_intent=anystring - write the text out to recovery.intent

61 * --update_package=root:path - verify install an OTA package file

62 * --wipe_data - erase user data (and cache), then reboot

63 * --wipe_cache - wipe cache (but not user data), then reboot

64 *

65 * After completing, we remove /cache/recovery/command and reboot.

66 */

67

68 static const int MAX_ARG_LENGTH = 4096;

69 static const int MAX_ARGS = 100;

70

71 // open a file given in root:path format, mounting partitions as necessary

72 static FILE*

73 fopen_root_path(const char *root_path, const char *mode) {

74 if (ensure_root_path_mounted(root_path) != 0) {

75 LOGE("Can't mount %s\n", root_path);

76 return NULL;

77 }

78

79 char path[PATH_MAX] = "";

80 if (translate_root_path(root_path, path, sizeof(path)) == NULL) {

81 LOGE("Bad path %s\n", root_path);

82 return NULL;

83 }

84

85 // When writing, try to create the containing directory, if necessary.

86 // Use generous permissions, the system (init.rc) will reset them.

87 if (strchr("wa", mode[0])) dirCreateHierarchy(path, 0777, NULL, 1);

88

89 FILE *fp = fopen(path, mode);

90 if (fp == NULL) LOGE("Can't open %s\n", path);

91 return fp;

92 }

93

94 // close a file, log an error if the error indicator is set

95 static void

96 check_and_fclose(FILE *fp, const char *name) {

97 fflush(fp);

98 if (ferror(fp)) LOGE("Error in %s\n(%s)\n", name, strerror(errno));

99 fclose(fp);

100 }

101

102 // command line args come from, in decreasing precedence:

103 // - the actual command line

104 // - the bootloader control block (one per line, after "recovery")

105 // - the contents of COMMAND_FILE (one per line)

106 static void

107 get_args(int *argc, char ***argv) {

108 if (*argc > 1) return; // actual command line arguments take priority

109 char *argv0 = (*argv)[0];
```

```
110

111 struct bootloader_message boot;

112 if (!get_bootloader_message(&boot)) {

113 if (boot.command[0] != 0 && boot.command[0] != 255) {

114 LOGI("Boot command: %.*s\n", sizeof(boot.command), boot.command);

115 }

116

117 if (boot.status[0] != 0 && boot.status[0] != 255) {

118 LOGI("Boot status: %.*s\n", sizeof(boot.status), boot.status);

119 }

120

121 // Ensure that from here on, a reboot goes back into recovery

122 strcpy(boot.command, "boot-recovery");

123 set_bootloader_message(&boot);

124

125 boot.recovery[sizeof(boot.recovery) - 1] = '\0'; // Ensure termination

126 const char *arg = strtok(boot.recovery, "\n");

127 if (arg != NULL && !strcmp(arg, "recovery")) {

128 *argv = (char **) malloc(sizeof(char *) * MAX_ARGS);

129 (*argv)[0] = argv0;

130 for (*argc = 1; *argc < MAX_ARGS; ++*argc) {

131 if ((arg = strtok(NULL, "\n")) == NULL) break;

132 (*argv)[*argc] = strdup(arg);

133 }

134 LOGI("Got arguments from boot message\n");

135 return;

136 } else if (boot.recovery[0] != 0 && boot.recovery[0] != 255) {

137 LOGE("Bad boot message\n\"%.20s\"\n", boot.recovery);

138 }

139 }

140

141 FILE *fp = fopen_root_path(COMMAND_FILE, "r");

142 if (fp == NULL) return;

143

144 *argv = (char **) malloc(sizeof(char *) * MAX_ARGS);

145 (*argv)[0] = argv0; // use the same program name

146

147 char buf[MAX_ARG_LENGTH];

148 for (*argc = 1; *argc < MAX_ARGS && fgets(buf, sizeof(buf), fp); ++*argc) {

149 (*argv)[*argc] = strdup(strtok(buf, "\r\n")); // Strip newline.

150 }

151

152 check_and_fclose(fp, COMMAND_FILE);

153 LOGI("Got arguments from %s\n", COMMAND_FILE);

154 }

155

156

157 // clear the recovery command and prepare to boot a (hopefully working) system,

158 // copy our log file to cache as well (for the system to read), and

159 // record any intent we were asked to communicate back to the system.

160 // this function is idempotent: call it as many times as you like.

161 static void

162 finish_recovery(const char *send_intent)

163 {

164 // By this point, we're ready to return to the main system...
```

```
165 if (send_intent != NULL) {
166 FILE *fp = fopen_root_path(INTENT_FILE, "w");
167 if (fp != NULL) {
168 fputs(send_intent, fp);
169 check_and_fclose(fp, INTENT_FILE);
170 }
171 }
172
173 // Copy logs to cache so the system can find out what happened.
174 FILE *log = fopen_root_path(LOG_FILE, "a");
175 if (log != NULL) {
176 FILE *tmplog = fopen(TEMPORARY_LOG_FILE, "r");
177 if (tmplog == NULL) {
178 LOGE("Can't open %s\n", TEMPORARY_LOG_FILE);
179 } else {
180 static long tmplog_offset = 0;
181 fseek(tmplog, tmplog_offset, SEEK_SET); // Since last write
182 char buf[4096];
183 while (fgets(buf, sizeof(buf), tmplog)) fputs(buf, log);
184 tmplog_offset = ftell(tmplog);
185 check_and_fclose(tmplog, TEMPORARY_LOG_FILE);
186 }
187 check_and_fclose(log, LOG_FILE);
188 }
189
190 // Reset the bootloader message to revert to a normal main system boot.
191 struct bootloader_message boot;
192 memset(&boot, 0, sizeof(boot));
193 set_bootloader_message(&boot);
194
195 // Remove the command file, so recovery won't repeat indefinitely.
196 char path[PATH_MAX] = "";
197 if (ensure_root_path_mounted(COMMAND_FILE) != 0 ||
198 translate_root_path(COMMAND_FILE, path, sizeof(path)) == NULL ||
199 (unlink(path) && errno != ENOENT)) {
200 LOGW("Can't unlink %s\n", COMMAND_FILE);
201 }
202
203 sync(); // For good measure.
204 }
205
206 #define TEST_AMEND 0
207 #if TEST_AMEND
208 static void
209 test_amend()
210 {
211 extern int test_symtab(void);
212 extern int test_cmd_fn(void);
213 extern int test_permissions(void);
214 int ret;
215 LOGD("Testing symtab...\n");
216 ret = test_symtab();
217 LOGD(" returned %d\n", ret);
218 LOGD("Testing cmd_fn...\n");
219 ret = test_cmd_fn();
220 LOGD(" returned %d\n", ret);
221 LOGD("Testing permissions...\n");
222 ret = test_permissions();
223 LOGD(" returned %d\n", ret);
224 }
```

```
225 #endif // TEST_AMEND
226
227 static int
228 erase_root(const char *root)
229 {
230 ui_set_background(BACKGROUND_ICON_INSTALLING);
231 ui_show_indeterminate_progress();
232 ui_print("Formatting %s...\n", root);
233 return format_root_device(root);
234 }
235
236 static void
237 prompt_and_wait()
238 {
239 ui_print("\n"
240 "Home+Back - reboot system now\n"
241 "Alt+L - toggle log text display\n"
242 "Alt+S - apply sdcard:update.zip\n"
243 "Alt+W - wipe data/factory reset\n");
244
245 for (;;) {
246 finish_recovery(NULL);
247 ui_reset_progress();
248 int key = ui_wait_key();
249 int alt = ui_key_pressed(KEY_LEFTALT) ||
ui_key_pressed(KEY_RIGHTALT);
250
251 if (key == KEY_DREAM_BACK &&
ui_key_pressed(KEY_DREAM_HOME)) {
252 // Wait for the keys to be released, to avoid triggering
253 // special boot modes (like coming back into recovery!).
254 while (ui_key_pressed(KEY_DREAM_BACK) ||
255 ui_key_pressed(KEY_DREAM_HOME)) {
256 usleep(1000);
257 }
258 break;
259 } else if (alt && key == KEY_W) {
260 ui_print("\n");
261 erase_root("DATA:");
262 erase_root("CACHE:");
263 ui_print("Data wipe complete.\n");
264 if (!ui_text_visible()) break;
265 } else if (alt && key == KEY_S) {
266 ui_print("\nInstalling from sdcard...\n");
267 int status = install_package(SDCARD_PACKAGE_FILE);
268 if (status != INSTALL_SUCCESS) {
269 ui_set_background(BACKGROUND_ICON_ERROR);
270 ui_print("Installation aborted.\n");
271 } else if (!ui_text_visible()) {
272 break; // reboot if logs aren't visible
273 }
274 ui_print("\nPress Home+Back to reboot\n");
275 }
276 }
277 }
278
279 static void
280 print_property(const char *key, const char *name, void
*cookie)
281 {
282 fprintf(stderr, "%s=%s\n", key, name);
283 }
284
285 int
```

```
286 main(int argc, char **argv)
287 {
288 time_t start = time(NULL);
289
290 // If these fail, there's not really anywhere to complain...
291 freopen(TEMPORARY_LOG_FILE, "a", stdout); setbuf(stdout, NULL);
292 freopen(TEMPORARY_LOG_FILE, "a", stderr); setbuf(stderr, NULL);
293 fprintf(stderr, "Starting recovery on %s", ctime(&start));
294
295 ui_init();
296 ui_print("Android system recovery utility\n");
297 get_args(&argc, &argv);
298
299 int previous_runs = 0;
300 const char *send_intent = NULL;
301 const char *update_package = NULL;
302 int wipe_data = 0, wipe_cache = 0;
303
304 int arg;
305 while ((arg = getopt_long(argc, argv, "", OPTIONS, NULL)) != -1) {
306 switch (arg) {
307 case 'p': previous_runs = atoi(optarg); break;
308 case 's': send_intent = optarg; break;
309 case 'u': update_package = optarg; break;
310 case 'w': wipe_data = wipe_cache = 1; break;
311 case 'c': wipe_cache = 1; break;
312 case '?':
313 LOGE("Invalid command argument\n");
314 continue;
315 }
316 }
317
318 fprintf(stderr, "Command:");
319 for (arg = 0; arg < argc; arg++) {
320 fprintf(stderr, " \"%s\"", argv[arg]);
321 }
322 fprintf(stderr, "\n\n");
323
324 property_list(print_property, NULL);
325 fprintf(stderr, "\n");
326
327 #if TEST_AMEND
328 test_amend();
329 #endif
330
331 RecoveryCommandContext ctx = { NULL };
332 if (register_update_commands(&ctx)) {
333 LOGE("Can't install update commands\n");
334 }
335
336 int status = INSTALL_SUCCESS;
337
338 if (update_package != NULL) {
339 status = install_package(update_package);
340 if (status != INSTALL_SUCCESS) ui_print("Installation aborted.\n");
341 } else if (wipe_data || wipe_cache) {
342 if (wipe_data && erase_root("DATA:")) status = INSTALL_ERROR;
343 if (wipe_cache && erase_root("CACHE:")) status = INSTALL_ERROR;
344 if (status != INSTALL_SUCCESS) ui_print("Data wipe failed.\n");
```

```
345 } else {

346 status = INSTALL_ERROR; // No command specified

347 }

348

349 if (status != INSTALL_SUCCESS) ui_set_background(BACKGROUND_ICON_ERROR);

350 if (status != INSTALL_SUCCESS || ui_text_visible()) prompt_and_wait();

351

352 // If there is a radio image pending, reboot now to install it.

353 maybe_install_firmware_update(send_intent);

354

355 // Otherwise, get ready to boot the main system...

356 finish_recovery(send_intent);

357 ui_print("Rebooting...\n");

358 sync();

359 reboot(RB_AUTOBOOT);

360 return EXIT_SUCCESS;

361 }
```

3. CONCLUSION

In this paper detail information on social engineering is described which provides a platform for learners. It describes all facts, methods and approaches followed or used to activate social engineering which could lead to an affective cause for cyber crime. It is being proved in an survey conducted at different organizations in Mysore city that about 60% of individual persons are affected by social engineering through the method of thrashing and about 40% of organizations lose their secure information through the method of thrashing. The result of the survey brings an awareness to almost all employees in the organization to be secure about their personal and important information from these hackers. Details on retrieving information and applying defragmentation is also given in the form of c code as a work carried out in a company Torus solutions, Mysore.

REFERENCES

[1] (Dr) P. Vinod Bhattathiripad , Cyber crime investigation consultant kerala. (http:// www. Saintgits.org/main/sie/computer%20science/annual%) 20 report. As per a seminar conducted at kerala on 7th august 2009 as given in annual report of the address mentioned above is the guidelines for basic ground work .

[2] http://download.cnet.com/Restoration/3000-2094_4-10322950.html

[3] Operating system concepts from galvino.

[4] www.torussolution.com a software firm in mysore,Karnataka.