New issue

Jump to bottom

HEAP OVERFLOW VULNERABILITY #22

Closed F1r opened this issue on Mar 19, 2019 · 4 comments

F1r commented on Mar 19, 2019

In CharCodeToUnicode.cc::264
!pst->getToken(tok3, sizeof(tok3), &n3) || //sizeof(tok3) == 256
here n3 can be a value from 0 to 256 - 2

In CharCodeToUnicode.cc::298
addMapping(code1, tok3 + 1, n3 - 2, i);

In CharCodeToUnicode.cc::313
if (n <= 4) {
if (sscanf(uStr, "%x", &u) != 1) {
error(-1, "Illegal entry in ToUnicode CMap");
return;
}
map[code] = u + offset;
} else {
if (sMapLen >= sMapSize) {
sMapSize = sMapSize + 16;
sMap = (CharCodeToUnicodeString *)
greallocn(sMap, sMapSize, sizeof(CharCodeToUnicodeString));
}
map[code] = 0;
sMap[sMapLen].c = code;
sMap[sMapLen].len = n / 4;

The parameter n of void CharCodeToUnicode::addMapping(CharCode code, char *uStr, int n, int offset) can be a bigger value than the the limited value maxUnicodeString
In CharCodeToUnicode.cc::350
sMap[sMapLen].u[sMap[sMapLen].len - 1] += offset;

Using the sample pdf file , we can find the VUL clearly.
sMapLen = 0xf
sMap = 0x6f56f0
pwndbg> p sMap[0xf]
$8 = {
c = 51,
u = {0, 17039378, 4784134, 7208965, 2686984, 14, 542, 185},
len = 12
}

pwndbg> p sMap[sMapLen].len
$9 = 12

so sMap[sMapLen].len - 1 = 13, which makes the array Unicode u[maxUnicodeString]; as follows oob write.
#define maxUnicodeString 8

struct CharCodeToUnicodeString {
CharCode c;
Unicode u[maxUnicodeString];
int len;
};

So, we can modify memory from offset 0 to 63 * 4 with type unsigned int by adding the original value with offset, which can still be controlled. Local command execution is possible using heap fengshui, especially in the linux machine using glibc version > 2.6. Free a chunk using the bigger fake size can lead to continuously heap buf overflow, which can make the hacker get a memory containing the function pointer and then achieve the purpose of command execution.

F1r commented on Mar 19, 2019 Author

In CharCodeToUnicode.cc::343

for (j = 0; j < sMap[sMapLen].len && j < maxUnicodeString; ++j) {
strncpy(uHex, uStr + j*4, 4);
uHex[4] = '\0';
if (sscanf(uHex, "%x", &sMap[sMapLen].u[j]) != 1) {
error(-1, "Illegal entry in ToUnicode CMap");
}
}

A better way to use the VULNERABILITY, we can precisely control the value to be written using sscanf.

F1r commented on Mar 20, 2019 Author

samples.zip

flexpaper commented on Mar 23, 2019 Owner

So to clarify; this would require a user to upload a malicious PDF file that contains an incorrect charcode map, correct?

flexpaper commented on Mar 23, 2019 Owner

Confirmed to be a requirement to have a malicious PDF file uploaded to the system for this vulnerability. Fixed by `80bf71f`

flexpaper closed this as completed on Mar 23, 2019

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

2 participants