网络安全实验报告

漏洞一：getenv("SCRIPT_NAME")

寻找过程

搜索程序中容易造成缓冲区溢出的函数 getenv()，结果如图 1 所示。

-> http.c 360　　dir_join(name, getenv("SCRIPT_NAME"), indices[i]);

函数 dir_join 的作用是将 dirname 复制到 dst，再把 filename 添加到 dst 末尾。由于 getenv 函数不会检查环境变量的长度，而 name 数组的长度为 1024bytes，所以 getenv(“SCRIPT_NAME”)的大小可能大于 1024 bytes，造成缓冲区溢出。

```
httpd@vm-6858:~/lab$ grep -n "getenv" *.c
http.c:242:    setenv("SCRIPT_NAME", pn + strlen(getenv("DOCUMENT_ROOT")), 1);
http.c:309:    if (getenv("PATH_INFO")) {
http.c:312:        snprintf(buf, 1024, "%s%s", pn, getenv("PATH_INFO"));
http.c:360:            dir_join(name, getenv("SCRIPT_NAME"), indices[i]);
zookfs.c:47:                http_serve(sockfd, getenv("REQUEST_URI"));
```

图 1 程序中所有的 getenv 函数位置

```
343  void dir_join(char *dst, const char *dirname, const char *filename) {
344      strcpy(dst, dirname);
345      if (dst[strlen(dst) - 1] != '/')
346          strcat(dst, "/");
347      strcat(dst, filename);
348  }
```

图 2 函数 dir_join 的实现

搜索环境变量“SCRIPT_NAME”的设置位置

-> http.c 242　　setenv("SCRIPT_NAME", pn + strlen(getenv("DOCUMENT_ROOT")), 1);

发现 SCRIPT_NAME 的值由 pn 和 DOCUMENT_ROOT 的大小组成，位于函数 split_path 中

在函数 split_path 中搜索局部变量 pn

-> http.c 196　　void split_path(char *pn)

发现 pn 是函数 split_path 的形参

搜索 split_path 的调用位置

-> http.c 283　　split_path(pn);

位于函数 http_serve 中

在函数 http_serve 中搜索局部变量 pn

-> http.c 279　　getcwd(pn, sizeof(pn));

-> http.c 282　　strcat(pn, name);

发现 pn 是由当前路径和 name 拼接而成

在函数 http_serve 中搜索局部变量 name
-> http.c 273 void http_serve(int fd, const char *name)
是函数 http_server 的形参

搜索 http_server 的调用位置
-> zookfs.c 47 http_serve(sockfd, getenv("REQUEST_URI"));
发现 name 的实参是环境变量 REQUEST_URI

搜索环境变量“REQUEST_URI”的设置位置
-> http.c 107 envp += sprintf(envp, "REQUEST_URI=%s", reqpath) + 1;
发现 REQUEST_URI 是 reqpath 设置的

在函数 http_request_line 中搜索局部变量 reqpath
-> http.c 64 const char *http_request_line(int fd, char *reqpath, char *env, size_t *env_len)
是函数 http_request_line 的传入形参

搜索函数 http_request_line 的调用位置
-> zookd.c 70 if ((errmsg = http_request_line(fd, reqpath, env, &env_len)))

搜索 reqpath 的定义位置
-> zookd.c 70 char reqpath[2048];
reqpath 最大长度为 2048，大于 1024，所以当 reqpath 大小大于 1024 时，会触发缓冲区溢出

漏洞触发

reqpath 是由客户端传入的请求解析获得的请求路径，所以需要修改客户端请求的路径，使其大于 1024 bytes 即可造成环境变量 SCRIPT_NAME 的大小大于 name 的大小，从而造成缓冲区溢出，所以修改 req = "GET /” +’A’*1024 “ HTTP/1.0\r\n\r\n"。

漏洞二：sprintf(envvar, "HTTP_%s", buf);

寻找过程

搜索程序中的 sprintf 出现的位置，如图 3 所示。
-> http.c 165 sprintf(envvar, "HTTP_%s", buf);
由于 envvar 大小为 512 bytes，而 buf 大小为 8192 bytes，所以当 buf 实际大小大于 512 bytes 就会造成缓冲区溢出。

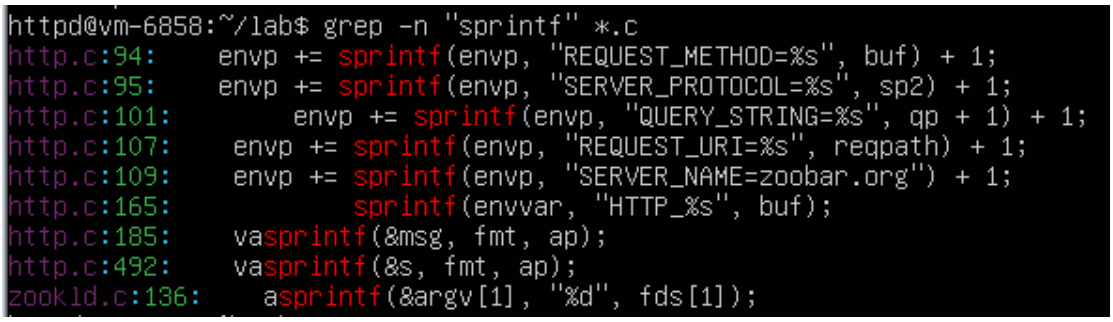

图 3 sprintf 出现位置

搜索局部变量 buf 的赋值位置

-> http.c 129 if (http_read_line(fd, buf, sizeof(buf)) < 0)

是函数 http_read_line 进行赋值的

搜索函数 http_read_line 的实现代码

-> http.c 36 int cc = read(fd, &buf[i], 1);

发现 buf 是从文件句柄 fd 读取的数据

搜索文件句柄 fd

-> http.c 116 const char *http_request_headers(int fd)

发现 fd 为函数 http_request_line 的形参

搜索函数 http_request_line 的调用位置

-> zookd.c 44 if ((errmsg = http_request_headers(sockfd)))

漏洞触发

envvar 存储 HTHP 头部选项信息，大小为 512bytes，所以只需要保证 Http 头部选项大小大于 512bytes 且保持正确的头部格式，即可造成缓冲区溢出，因此可以将 req 改为：

req ="GET / HTTP/1.0\r\nA: " + 'A'*532 + "\r\n\r\n"

实验结果

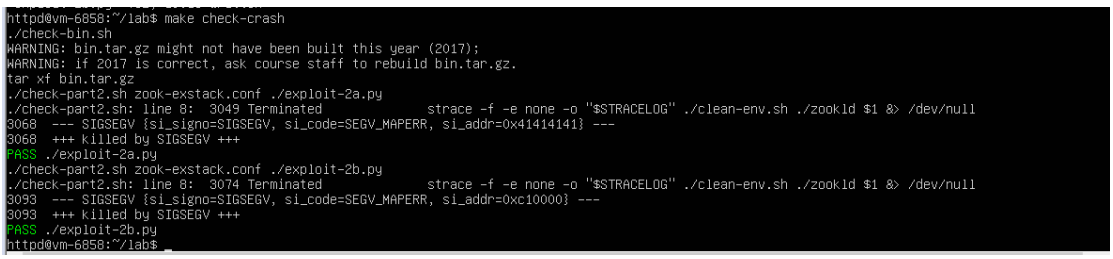

图 4 实验结果