进程监控SDK接口文档

|  |  |
| --- | --- |
| 版本号 | Ver\_1.2.0 |
| 制定 | 江民Linux研发组 |
| 日期 | 2018.2.27 |

SDK组成：

proc\_monitor.h

libnet\_monitor.so

1. 监控初始化
2. 回调函数

typedef enum network\_protocol

{

NP\_UNKNOWN = 0,

NP\_TCP,

NP\_UDP,

NP\_HTTP,

NP\_HTTPS,

NP\_DNS,

} network\_protocol\_t;

typedef void (\*net\_monitor\_callback\_t)(network\_protocol\_t protocol, void \*p\_info, void \*p\_user\_data);

说明：

监控过程中的回调函数。

注意，不要在本函数中有耗时的操作；同时，回传的数据会在函数结束后失效，如需保存数据，请自行拷贝。

参数：

protocol：本次事件的标识

NP\_DNS DNS信息

NP\_HTTP HTTP信息

其它暂未使用

p\_user\_data：定义回调函数的用户所使用的数据。

p\_info: 一个关于网络信息的结构体，需要根据protocol的类型进行强制类型转换，定义如下：

typedef struct conn\_info

{

char source\_address[16]; // 源IP， 强转为一个in\_addr\_t \*，然后取值

char destination\_address[16]; // 目的IP，强转为一个in\_addr\_t \*，然后取值

uint16\_t source\_port; // 源端口，网络字节序

uint16\_t destination\_port; // 目的端口，网络字节序

}conn\_info\_t;

typedef struct dns\_info // 承载DNS信息的结构体

{

conn\_info\_t connection;

const char \*p\_dns\_query; // DNS请求的域名

char response\_address[46]; // DNS响应的IP，点分十进制表示

time\_t time; // 发生记录的时间

}dns\_info\_t;

typedef struct network\_info // 承载HTTP信息的结构体

{

conn\_info\_t connection;

network\_protocol\_t protocol; // 网络协议

const char \*p\_url; // 请求的URL

time\_t time; // 发生请求的时间

uint8\_t result; // 未使用

}network\_info\_t;

返回：

暂定void，将来可能会根据是否拦截改为bool类型（比如，返回true放行，false拦截）

1. 初始化接口

typdedef void proc\_monitor\_t;

net\_monitor\_t \*init\_net\_monitor(net\_monitor\_callback\_t callback, void \*p\_user\_data, size\_t option\_count, const char \*(\*pp\_options)[2]);

参数：

callback：发生事件时的回调函数。

p\_user\_data：回调函数的p\_user\_data参数。

option\_count: 监控选项的个数

pp\_options: 是一个二维数组，表示的监控选项。pp\_options有option\_count个元素，每个元素均由一对const char \*组成key-value的形式。其中，(\*pp\_option)[0]为key，(\*pp\_option)[1]为value。具体的内容参考[设置参数](#设置参数)一节。

返回：

成功初始化返回非空指针，否则返回NULL。

说明：

初始化进程监控

1. 设置参数

bool set\_option(proc\_monitor\_t \*p\_monitor, const char \*p\_option\_name, const char \*p\_option\_value);

说明：

设置option，或改变option。SDK内部会依据option的名称，自动将p\_option\_value转换到实际的类型。

参数：

p\_option\_name:option的名称。

p\_option\_value:option的值。当实际的类型不是char \*时，需要强制类型转换为char \*。

返回：

设置成功返回true，否则返回false。

可设置参数：

|  |  |  |
| --- | --- | --- |
| 选项名称 | 参数类型 | 默认值 |
| enable\_http  是否启用HTTP监听 | int32\_t | 0 |
| enable\_dns  是否启用DNS监听 | int32\_t | 0 |

1. 查询参数

const char\* get\_option(proc\_monitor\_t \*p\_monitor, const char \*p\_option\_name);

说明：

返回的option的值。

参数：

p\_option\_name:要查询的option名称。可选值参考[设置参数](#设置参数)。

返回：

返回option对应的值。当实际的类型不是char \*时，需要用户做类型转换。

出错则返回NULL。

1. 停止监控

void fin\_proc\_monitor(proc\_monitor\_t \*p\_proc\_monitor);

当程序退出时，调用本接口终止对进程的监控，并释放相关资源。

1. 使用实例

#include <stdio.h>

#include <inttypes.h>

#include <unistd.h> // sleep

#include <arpa/inet.h> // inet\_ntop, ntohs

#include "net\_monitor.h"

static void net\_monitor\_call\_back(network\_protocol\_t protocol, void \*p\_info, void \*p\_user\_data)

{

if(protocol == NP\_HTTP) {

network\_info\_t \*p\_network\_info = (network\_info\_t \*)p\_info;

printf("%s\n", "NP\_HTTP");

printf("\turl:[%s]\n", p\_network\_info->p\_url);

}

else if(protocol == NP\_DNS) {

dns\_info\_t \*p\_dns\_info = (dns\_info\_t \*)p\_info;

printf("%s\n", "NP\_DNS");

printf("\tquery:[%s] response:[%s]\n",

p\_dns\_info->p\_dns\_query,

p\_dns\_info->response\_address);

}

else {

printf("%s\n", "NP\_UNKNOWN");

return;

}

// tuple

conn\_info\_t \*p\_conn = (conn\_info\_t \*)p\_info;

char src\_ip\_str[16], dst\_ip\_str[16];

inet\_ntop(AF\_INET, p\_conn->source\_address, src\_ip\_str, sizeof(src\_ip\_str));

inet\_ntop(AF\_INET, p\_conn->destination\_address, dst\_ip\_str, sizeof(dst\_ip\_str));

printf("\ttuple [%s]:[%d] --> [%s][%d]",

src\_ip\_str, ntohs(p\_conn->source\_port),

dst\_ip\_str, ntohs(p\_conn->destination\_port));

putchar(10);

}

void wait\_for\_exit()

{

printf("exit with %c\n", getchar());

}

int32\_t main(int32\_t argc,char \*args[])

{

int32\_t enable\_http = 1;

int32\_t enable\_dns = 1;

const char \*options[][2] = { {"enable\_http", (const char \*)&enable\_http },

{"enable\_dns", (const char \*)&enable\_dns }

};

net\_monitor\_t \*handle = init\_net\_monitor(net\_monitor\_call\_back, NULL, sizeof(options)/sizeof(options[0]), options);

sleep(10);

enable\_http = 0;

set\_option(handle, "enable\_http", (const char \*)&enable\_http);

wait\_for\_exit();

fin\_net\_monitor(handle);

}

1. 部署

// TODO

描述如何安装。

依赖GLIBC\_2.2.5、CXXABI\_1.3.10。