I. INTRODUCTION

Multiple trojans have been implemented to address each of the three general categories of attacks, which include functional specification modifications, information leakage and denial of service. Section II first provides the team’s analysis of the encryption device and possible attack vectors. Section III details a trojan which transmits unencrypted message text in real-time as it is typed. The second trojan provides an adversary the ability to obtain the master encryption key, and is described in Section IV. Section V details a Denial Of Service (DoS) attack that disables the ability to transmit encrypted text over the RS232 port. Section VI addresses the last trojan, a method of strobing key presses as they occur over LEDs.

II. ANALYSIS AND ATTACK VECTORS

In designing trojans for the Alpha encryption device, contest restrictions, in addition to the standard device operation, were both taken into account. Although some attacks theorized by the team may have been less trivial to detect, they were not implemented if difficult to verify without additional hardware (per contest rules), or unlikely to be carried out successfully by an adversary in the setting in which the device was intended to be used.

The Alpha device requires that an attacker have physical access to the device and either a PC, or a device capable of entering keystrokes via the PS/2 port, and retrieving data over the serial port. Therefore, it is desirable that the trojans provide an attacker the means to perform actions quickly, without arousing suspicion through the use of extraneous hardware. As a result, the trojans implemented and presented in this document utilize serial communications to leak information, and are enabled via keyboard input.

The aforementioned restrictions eliminated some of the trojans theorized by the team. For example, simple observation of the device’s functionality suggests that the red and blue pins of the RGB connector were left unused. This provides the opportunity to leak sensitive information over one of these unused pins. However, this would require that additional hardware be constructed to monitor these pins. Additionally, this attack would include the challenge of leaking information while not interfering with the information displayed on the monitor. Carrying out this attack would be highly valuable if an attacker were able to place a malicious monitor, which would include the hardware necessary to read and store information leaked on the unused pins of the RGB connector, in the location where the Alpha device was being operated. Security-conscious users may suspect the use keystloggers connected to a keyboard, but would be far less likely to suspect that a monitor would be logging information. Due to time restrictions, this attack was not further investigated.

Since standard operation of the device requires that information transmitted over the RS232 connection be validated during testing, the trojans described in sections III and IV were designed to output sensitive information via a second “hidden” serial port. As a proof of concept, this port is connected to the JD 6-pin header on the Basys board, as shown in Figure 1. However, this “hidden” serial transmitter could be connected to other unused pins to avoid detection during testing.

Figure 1: Alpha Setup With "Hidden" Transmitter

Upon reviewing the code for the Alpha project, the team found that the master key was hard-coded in alphatop.v. Furthermore, it was noted that the eight “key” bits set by the user prior to performing encryption are included in the encrypted message. The packet format includes the 8-bit modifier for the 128-bit master key, enabling the provided enc_verifier program to automatically decrypt any message with a matching master key. That is, any 8-bit value specified does not need to be given to the enc_verifier program; it is given already. Therefore, by obtaining the master key, an adversary could easily decrypt any intercepted messages. This observation has motivated the development of the trojan described in Section IV.

Although the Alpha device provides a means to encrypt sensitive data prior to an exchange, the level of security the device provides can only be obtained through correct
operation. As shown through numerous incidents within the past decade, vulnerabilities often arise due to human negligence in the configuration or operation of a security system. Therefore, if a system can be compromised in a manner which forces users into patterns of misuse, further opportunities for an attacker can be created. The team noted that decreasing the reliability of the Alpha device may cause its users to simply revert to sending plaintext messages, rather than using the device to encrypt messages. This belief has motivated the Denial of Service attack addressed in Section IV.

III. LEAKING KEY-PRESSES

An attack that leaks a plaintext message is very advantageous to an attacker, since it circumvents the time and effort needed to crack or decrypt a message. To do this, the simplest of backdoors was implemented. An asynchronous transmitter was inserted to be used at I/O header D, as to avoid in band detection on the primary serial port, as shown in Figure 1. If the enc_verifier program was instead designed to be stricter, out of band data could be detected, and the trojan would be exposed. However, using another serial port doesn’t modify the power usage, and cannot be detected unless it’s being explicitly checked for, which is unlikely if the attacker chooses to use output pins that are less likely to be monitored, as the I/O jumpers chosen for this proof of concept most likely would be.

To leak key-presses over the serial port, there are two distinct possibilities. One could leak them as they happen, or, conversely, leak them later via a buffer, skipping the encryption step. In the example design, the former was chosen as the simpler, more reliable design. This requires no additional storage, and is produced in real-time, as opposed to at a later period. To do this, the keyboard output is connected directly to the input of the serial transmitter module, with a valid signal driving the enable signal of the transmitter. The result of this is that whenever a key is pressed, it is immediately sent over the “hidden” serial port. The latter approach would allow an attacker to retrieve plaintext without requiring a means to monitor it as a user types. Although this would be a more elusive attack, it would also be more difficult to hide at the development level, as an additional FIFO was found to increase the slice usage significantly.

To view the leaked key presses, either the modified enc_verifier program listed in Appendix A or HyperTerminal can be used.

IV. LEAKING MASTER ENCRYPTION KEY

If an attacker is capable of intercepting messages encrypted with the Alpha device, a need to decrypt or crack the message(s) arises. As noted in Section II, the message format containing the 8-bit encryption modifier allows an attacker to simply decrypt a message, given the master key and knowledge of how the modifier is applied. An attacker could potentially reverse engineer the modifier, given knowledge of the master key, and the opportunity to perform sufficient iterations to deduce how the master key is manipulated prior to encryption. Furthermore, as the crypto team pointed out in alphatop.v, there are only 8-bits of entropy available. This means that if the master key is leaked (as opposed to the modified key) the base to which one can get the other encryption keys is available. This gives only 256 possible combinations to test: a trivial exercise even with the most modest of computing resources available. Therefore, armed with the master key, an attacker should be able to decrypt any intercepted message.

It is not sufficient to simply hardcode the master key to output, as this version of Alpha is a “pre-production” release and it may not be the same as the final release. As a result, the team chose to output bytes from the master_key wire directly. A separate module (pulse_key.vhd) was added to perform this task. Note that for ease of testing and development, this was implemented in a separate file. It would be in the best interest of an attacker to ensure that added modules be less conspicuous. The pulse key module outputs a single byte of the master key with each rising clock edge, which occurs each time the “trigger” scan code is received. Although data is sent to the “hidden” serial port, this design could be further developed to leak data over a blinking power LED or unused RGB pin (as mentioned in Section II).

In the provided example implementation of this trojan, the F6 key was used to trigger the output of a byte of the master key. This implies that it takes 16 keyboard presses to fully send the key over the hidden serial port. This was chosen as the most reliable vector for distribution of the key, since the attack can be performed quickly, without arousing a great deal of suspicion. If this attack is combined with the trojan described in Section III, an attacker could effectively obtain the master key when placing or retrieving a device used to log the plaintext key presses.

Figure 2: Leaking plaintext key strokes over hidden RS232 transmitter
In Appendix A, a modified enc_verifier program used to dump the raw hex bytes sent to the serial port is listed. This enables an attacker to see the master key dumped in hexadecimal format. Note that this program simply monitors the “hidden” serial port, so if plaintext keystrokes are entered, they will be included in the captured data, in addition to the master key as it’s leaked through the repeated F6 keystrokes.

**Figure 3: Master Key Leaking Trojan**

V. Disabling Transmit

In order to degrade the performance of the *Alpha* device, the team chose to disable the serial transmitter located on pin 4 of the 6-pin header labeled “JA” (See Figure 4). More subtle performance degradation could be achieved through modifying encrypted text, or the 8-bit “key” used to perform the encryption. However, the team felt that the inability to transmit encrypted text could certainly drive a user to forego the use of the *Alpha* device, while remaining an easily verifiable proof of concept.

The F6 key on the keyboard is used as a “trigger” for the Denial of Service trojan. A wire connecting the kb2ascii module to the alphatop module was added to denote the activation status of the trojan. An additional case for the trigger key (F6) was added to the large list of scan code cases in the kb2ascii module, where the team felt a few additional entries would be less noticeable than in other modules.

Although the “trigger” for the trojan was chosen to be the F6 key for the submitted code, slightly more elaborate conditions for the activation of the denial of service attack could be implemented. If the F6 key were replaced with a less commonly used scan code, such as one used by a keyboard with multimedia keys, a microcontroller embedded in a “malicious keyboard” could be used to randomly send the triggering scan code, resulting in what would appear to be intermittent failures of the *Alpha* device. Although not implemented, the team noted that unused keys such as Caps Lock could be added to the case block in the kb2ascii module. These extra “unused” keys would cause typographical errors that would typically have no effect, to disable the transmitter. To avoid detection, a trojan utilizing unused keys could be further developed to disable the transmitter after these keys have been hit a variable number of times. The number of key presses required to disable the transmitter could be initialized to data determined by the user, such as the least significant four bits of the first scan code provided. Although this would reduce the risk of a user associating erroneous keystrokes with the failure symptoms, it could increase the risk of detection after synthesis due to the inherent increase in the gate count. As a result, the team chose to avoid creating any sort of keystroke counter for the provided DoS trojan.

As implemented, the trojan used to degrade the performance consumes a minimal number of extra gates, compared to the aforementioned possible additions. The team believes that coupled with a “malicious keyboard” used to send a triggering scan code, this attack could be carried out successfully, causing the device to appear to fail intermittently. If this rate of failure could be controlled to avoid suspicion, yet still maintain a frequency that significantly inconveniences users, it is possible that users would forego the use of the *Alpha* device in order to maintain the status quo of their workflow.

**Figure 4: DoS Trojan Disabling RS232 Tx pin**
VI. STROBING ASCII VALUES

To allow the attacker to know what is being typed into Alpha remotely a different approach had to be taken to implementing a transmission mechanism. To achieve this, a series of five LEDs were used to represent ASCII letter values. Five bits is less than what is required to transmit ASCII values in full, however due to implementation constraints and making it so that the backdoor is not as obvious letters are capitalized (to save one bit) and numeric values are omitted, which brings the character set to a total of 26. To output the five bit vector the four dots on the four-seven-segment display are used in conjunction with the middle bar in the right most seven segment display. The middle bar was chosen in order to avoid easy detection by making it seem that the number changing from zero to eight is part of the operation of Alpha. Values are strobed to the output mechanism as keys get pressed. The time that the LEDs remain on is relatively short for the human eye to pick up but enough for a fast camera to detect. If the “under handing” operator is willing to risk detection he may enable the strobed values to last all the way until a new value is pressed and received from the keyboard. The output is simple to understand being that it’s just a character’s ASCII value without the leading three bits from its eight bit (or seven regarding ANSI) being displayed. The values being strobed to the dots on the seven segment displays represent the four MSBs of the five bit value and the bar going across the middle of the last seven segment display represents the LSB.

VII. CONCLUSION

As described above, separate trojans have been implemented to cover a range of possible attacks. In order to attempt to maintain verifiability, in addition to reducing their footprint, these trojans have been designed to be fairly simple. These trojans are included in separate builds, with the folder/zip file name denoting the contents of the build. Based upon these trojans, far more elaborate and elusive attacks could be derived, given that an attacker has sufficient time to minimize the footprint of the trojans.
Appendix A - Modified enc_verifier program used to obtain leaked master key

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>

#define BAUDRATE B9600
#define MODEMDEVICE "/dev/ttyS0"
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;

int main()
{
  int fd,c,rx,i;
  struct termios oldtio,newtio;

  fd = open(MODEMDEVICE, O_RDONLY | O_NOCTTY );
  if (fd <0) {perror(MODEMDEVICE); exit(-1); }

  tcgetattr(fd,&oldtio); /* save current port settings */
  bzero(&newtio,sizeof(newtio));
  newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
  newtio.c_iflag = IGNPAR;
  newtio.c_oflag = 0;
  /* set input mode (non-canonical, no echo,...) */
  newtio.c_lflag = 0;
  newtio.c_cc[VTIME] = 0;   /* inter-character timer unused */
  newtio.c_cc[VMIN]  = 1;   /* blocking read until 5 chars received */
  tcflush(fd, TCIFLUSH);
  tcsetattr(fd,TCSANOW,&newtio);

  printf("Waiting for transmission to begin.....\n");
  while (1) {/* loop for input */
    unsigned char buf[8];
    int i=0;
    int e=0;
    rx = read(fd,buf,8);
    if(rx<0) {printf("Error %d", rx);exit(-1);}
    fflush(stdout);
    for(i=rx;i > 0;i--) {
      printf("%02x",buf[rx - i]);
      fflush(stdout);
    }
    bzero(buf,sizeof(buf));
  }
}
```
tcsetattr(fd,TCSANOW,&oldtio);
close(fd);
}