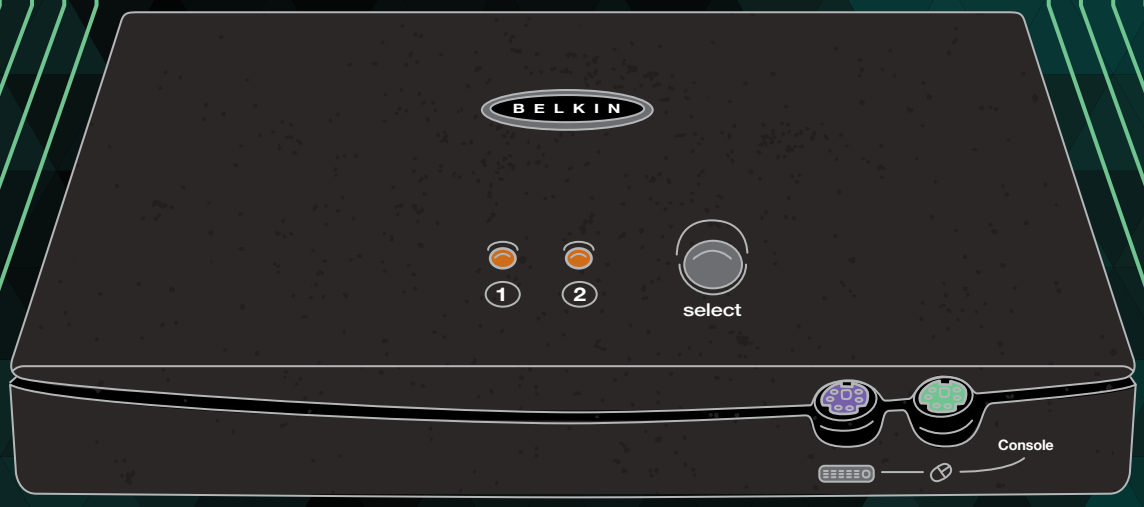
H A C K I N G

THE BELKIN E SERIES OMNIVIEW 2-PORT KVM SWITCH

BY IAN PAYTON, SECURITY ADVISORY EMEAR



TALOS

INTRODUCTION

Too frequently security professionals only consider software vulnerabilities when considering the risks of connecting devices to their networks and systems. When it comes to considering potential risks of connected devices and the Internet of Things, not only must security professionals consider potential vulnerabilities in the software and firmware of these systems, but also physical vulnerabilities in hardware.

Tampering with hardware is method by which attacker can physically modify systems in order to introduce new malicious functionality, such as the ability to exfiltrate data without resorting to exploiting software based vulnerabilities.

In this paper, we demonstrate the possibility of modifying a standard KVM switch to include an Arduino based key logger. We show that this can be achieved using off-the-shelf tools and components by anyone with a minimum of electronic engineering and programming knowledge.

KVM switches are hardware devices frequently used in operational environments that allow a user to easily switch between and control multiple computers from a single keyboard, monitor, and mouse.

They fall into some broad categories:

- Entry-level domestic and SoHo KVM switches operating from a physical button with little scope for hacking.
- 'Hotkey' KVM switches that allow the user to switch between attached computers by entering a combination of key presses. The inclusion of a microcontroller to identify a hotkey press suggests that these devices may be subverted as a key-logger.
- Enterprise level KVM switches offering tighter system integration. These are likely to be significantly more complex and may be running small, real-time operating systems with the consequent opportunities for hacking.

Here we describe the analysis of a KVM switch from the second category. Our choice was inspired by a client engagement where a client noticed an RJ45 port on their KVM switch and asked us to assess the security of the device.

The Belkin E Series OmniView 2-Port KVM switch is a domestic/SoHo unit that provides hotkey switching. This was selected as being representative of devices in this category so the outcome of any analysis is expected to be broadly applicable across similar devices from other manufacturers. Low cost units can be found on eBay (less than £10).

© 2017 Cisco. All rights reserved.

Belkin E Series OmniView 2-Port KVM Switch

OPENING THE CASE

Inside the case a set of components is revealed, including the following:

MICROCONTROLLER PIC16C57C

- This is an OTP PIC microcontroller made by Microchip Technology. This can be seen in the picture below as the larger chip towards the right of the PCB at the rear next to the cylindrical black buzzer.
- As can be seen from the picture, the microcontroller is in a DIP package and is mounted in a socket. This makes it particularly easy to remove the microcontroller to help with reverse engineering.

5 X 74HC4053D

- These are triple analogue dual multiplexers made by NXP. A dual multiplexer is capable of switching a single input to one of two outputs so these five devices likely form the core logic for switching the PS/2 keyboard and mouse signals between one of the four output ports.

© 2017 Cisco. All rights reserved.

THE PIC16C57C MICROCONTROLLER

This microcontroller is from a family of popular microcontrollers made by Microchip Technology. These microcontrollers are also popular in the hobby market, so there is a large amount of documentation readily available. The datasheet for the PIC16C57C can be found amongst a range of documentation on the microcontroller family here:

- http://www.microchip.com/wwwproducts/en/PIC16C57C

This includes documentation on the programming and verification protocol, which is useful for reverse engineering.

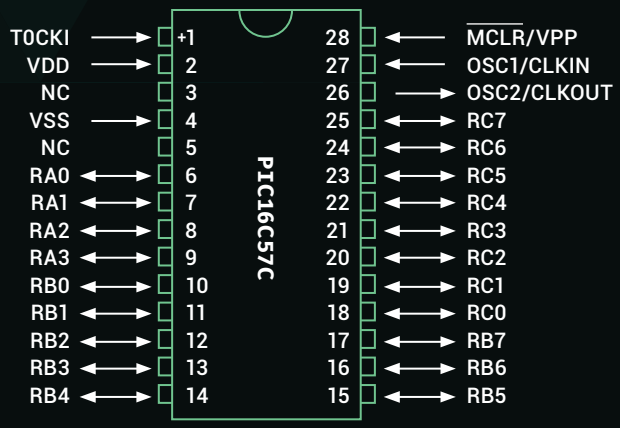
Belkin E Series OmniView 2-Port KVM Switch Internals

FIRMWARE ANALYSIS

PROGRAMMING AND VERIFICATION PROTOCOL

The PIC16C57C Programming/Verify specification outlines how to perform a 'quick verify' procedure on the microcontroller. This involves the following:

1. Apply power (5V Vdd, Ground Vss).
2. Hold T0CKI high.
3. Hold OSC1 low.
4. Bring Vpp up to the programming voltage (13V) (this puts the PIC into programming mode and resets the program counter).
5. The value at the current program counter can be read from pins RA0-RA3, RB0-RB7 (the PIC16C57C has a 12-bit word).
6. Set OSC1 high. This increments the program counter.
7. Set OSC1 low and repeat step 5.
8. Continue until all locations are read.



Using this procedure, the first location to be read is a special configuration register which has a pseudo-address of 0xFFF. Once OSC1 is clocked (at steps 6 and 7) the next location to be read is memory location 0x000 then location 0x001 and so on.

© 2017 Cisco. All rights reserved.

BUILDING A VERIFIER

Commercial PIC programmers can be used to read the contents of the PIC using this procedure, but it is also a sufficiently straight-forward protocol that a simple single-purpose 'PIC16C57C Verifier' can be created with a small microcontroller development board such as an Arduino (any similar system with a sufficient number of easily available general purpose I/O pins could be used - such as a Raspberry Pi). The 13V programming voltage needs to be applied externally, as development systems such as Arduino and Raspberry Pi tend to only have 5V and other low voltages available.

The picture on the right shows an Arduino Uno board wired to a ZIF socket holding the PIC16C57C removed from the KVM switch. The power supply for providing the 13V programming voltage is also shown. Due to I/O restrictions on the Arduino Uno, only four bits are being read from the PIC but it would be possible to reconfigure the device to read all other bits, such that the entire contents of the PIC16C57C could be read in several passes. Other development boards (such as the Arduino Mega) have sufficient I/O to read all 12 bits simultaneously.

CODE PROTECTION

The first word to be read from the PIC16C57C is the configuration register. This contains configuration data for the watchdog and oscillator, and also contains the 'code protection' bit. If the 'code protection' bit is zero, then code protection is enabled. When code protection is enabled, it is not possible to read the contents of the PIC16C57C memory (the verification operation succeeds, but the value returned does not represent the valid contents of the associated PIC16C57C memory location).

Unfortunately, reading the configuration register from the PIC16C57C taken from the Belkin KVM switch showed that the code protection bits were enabled, meaning that it was not possible to read the firmware from the PIC16C57C.

Verification of PIC16C57C using Arduino

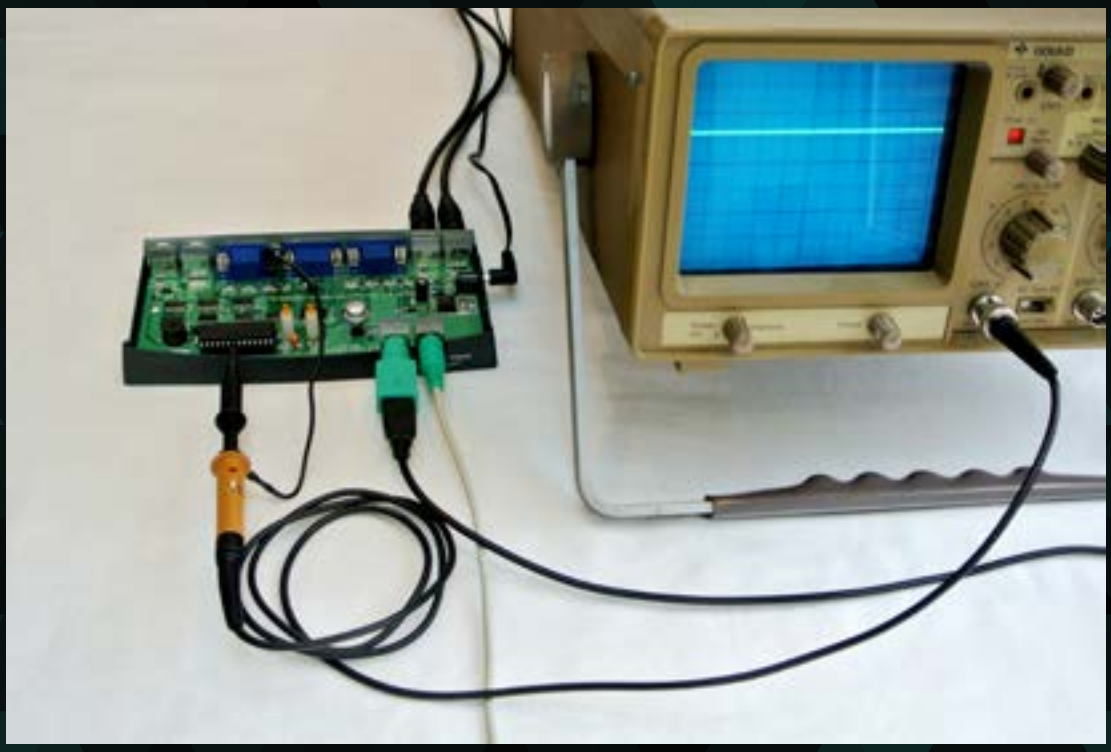
LOGIC ANALYSIS

RATIONALE

The previous sections detail the firmware analysis procedure, with the conclusion that the firmware cannot be read from the PIC16C57. This means that the goal of subverting the KVM switch for use as a key-logger must be achieved in one of two other ways:

- Analyse the logic implemented by the PIC16C57C microcontroller and rewrite the firmware from scratch on an equivalent PIC device.
- Analyse enough of the logic implemented by the PIC16C57C microcontroller to determine how to piggyback a secondary microcontroller to a subset of the device pins in order to monitor keypress data and implement a key-logger in the secondary microcontroller.

© 2017 Cisco. All rights reserved.



An oscilloscope being used to investigate pin functionality while keys are pressed on a keyboard.

DETERMINE PINOUT FUNCTIONS

OVERVIEW

The PIC16C57C has 20 general purpose I/O pins. The function of these pins can be investigated by using a multimeter, oscilloscope, or logic analyser on the pin. This should be performed while putting the KVM switch through various scenarios in order to see how the pin behaviour correlates with the functionality of the KVM switch. Voltage transitions on the pins may be slow (e.g. corresponding to an LED turning on or off) or fast (e.g. corresponding to the clock/data lines of the keyboard or mouse interfaces). Fast transitions might only be detected by an oscilloscope or logic analyser.

Each pin should be investigated while running through a variety of functional scenarios with the KVM switch:

- Switching between outputs.
- Typing on the keyboard while each of the outputs is active.
- Moving the mouse while each of the outputs is active.
- Plugging an end system in/out of one of the output ports.

Further analysis of the PCB layout can be made with the 'resistance' mode of a multimeter to determine which pins are directly connected to other components on the PCB. This is made easier by the fact that the PIC16C57C in the Belkin KVM switch is mounted in a socket, meaning that removing the PIC from the socket allows resistance between pins and other components to be more reliably measured.

© 2017 Cisco. All rights reserved.

Some of the pins on the PIC16C57C have fixed functionality (see pinout diagram on page 3) but it is the general purpose I/O pins 6-25 which are of interest. Investigation with an oscilloscope and multimeter revealed the following:

PINS 6-14

Pin 6 - RA0: Usually high, but pulled low while push-button switch is depressed.

- *Most likely the input pin for the push-button switch to change outputs.*

Pin 7 - RA1: Low when KVM Port 1 output is selected and high when KVM Port 2 output is selected.

- *Examining traces on the PCB, this also matches the sense of the input select pins on the 74HC4053D multiplexers.*
- *Using the multimeter in resistance mode, this pin is connected to the multiplex selector inputs S1, S2 and S3 on ICs U7, U8, U9, U10 (the 74HC4053D multiplexers).*
- *This therefore looks like an output pin that selects between Port 1 and Port 2 output on the KVM switch.*

Pin 8 - RA2: As with Pin 7, this is low when KVM Port 1 output is selected and high when KVM Port 2 output is selected.

- *Using the multimeter in resistance mode, this pin is connected to the multiplex selector inputs S1, S2, and S3 on IC U2*
- *Using the multimeter in resistance mode, this pin is also connected to OE1 in IC U3 (similar functionality to the multiplexer).*
- *This therefore looks like another output pin that selects between Port 1 and Port 2 output on the KVM switch.*
    - *It's not clear why both Pin 7 and Pin 8 appear to have similar functionality. One possibility is that Pin 7 controls the PS/2 (mouse, keyboard) switching, and Pin 8 controls the video switching. This would need further investigation.*

Pin 9 - RA3: Usually high, but activity is seen during switching between ports on the KVM switch.

- *Looking in detail at the activity on this pin during switching, the oscilloscope showed a waveform of about 3 cycles/cm on the oscilloscope screen when the scope is on 1ms/cm time-base - which makes this a 3kHz waveform.*
- *It is likely that this pin is directly driving the buzzer as a short high pitched (about 3kHz) beep is made by the KVM switch under certain circumstances, for instance, when switching between output ports.*

Pin 10 - RB0 - High when there is a device connected to KVM Port 1.

- *This appears to be an 'output enable' for the device connected to Port 1.*
- *During switching between ports, this pin is pulled low for a significant period (~1 sec) and then returns high.*
- *When the pin is pulled low, no output is seen from Port 1. There is activity on pins 20-23 (mouse/keyboard).*
    - *As a point of interest, the LEDs showing which port is selected also do not change until after this ~1 sec period).*

Pin 11 - RB1: High when there is a device connected to KVM Port 2.

- *This appears to be an 'output enable' for the device connected to Port 2.*
- *During switching between ports, this pin is pulled low for a significant period (~1 sec) and then returns high.*
- *When the pin is pulled low, no output is seen from Port 2. There is activity on pins 20-23 (mouse/keyboard).*
    - *The LEDs showing which port is selected also do not change until after this ~1 sec period).*

12 - RB2 - HIGH: No activity seen, no connections seen on PCB.

13 - RB3 - LOW: No activity seen, no connections seen on PCB.

14 - RB4 - HIGH: No activity seen.

- *PCB trace visible, apparently to Pin 14 on U9. Confirmed with multimeter in resistance mode.*
- *This is 'Input 1' on the 74HC4053 which means it will be switch between a pin on KVM Port 1 and Port 2 via the multiplexer.*
- *This may be an input or output — that is, either the KVM switch will send a signal to the PS/2 connector on the KVM output port, or receive a signal. However, no activity is seen on this pin in any of the scenarios tested.*

© 2017 Cisco. All rights reserved.

PINS 15-25

Pin 15 - RB5 - HIGH: No activity seen, no connections seen on PCB.

Pin 16 - RB6 - HIGH: No activity seen, no connections seen on PCB.

Pin 17 - RB7 - HIGH: No activity seen, no connections seen on PCB.

Pin 18 - RC0 - LOW: No activity seen, no connections seen on PCB.

Pin 19 - RC1 - HIGH: No activity seen, no connections seen on PCB.

Pin 20 - RC2: PS/2 Mouse – Clock

- *The oscilloscope shows regular bursts of pulses on this pin when the mouse is moved.*
- *The regularity of the pulses implies that this is the PS/2 clock for the mouse.*
- *Some activity on this pin was also seen during switching between KVM ports, when there was no mouse movement. See comments below about the purpose of pins 10 and 11.*

21 - RC3: PS/2 Mouse – Data

- *The oscilloscope shows irregular bursts of pulses on this pin when the mouse is moved.*
- *The irregularity of the pulses implies that this is the PS/2 data for the mouse.*
- *Some activity on this pin was also seen during switching between KVM ports, when there was no mouse movement. See comments below about the purpose of pins 10 and 11.*

22 - RC4: PS/2 Keyboard – Clock

- *The oscilloscope shows a regular burst of pulses on this pin when a key is pressed or released.*
- *The regularity of the pulses implies that this is the PS/2 clock for the keyboard.*

23 - RC5: PS/2 Keyboard – Data

- *The oscilloscope shows an irregular burst of pulses on this pin when a key is pressed or released.*
- *The irregularity of the pulses implies that this is the PS/2 data for the keyboard.*

24 - RC6: LED1

- *This pin corresponds directly to the state of the LED for KVM Port 1. It is therefore likely to be the output driver for the LED.*

25 - RC7: LED2

- *This pin corresponds directly to the state of the LED for KVM Port 2. It is therefore likely to be the output driver for the LED.*

© 2017 Cisco. All rights reserved.

PURPOSE OF PINS 10-11: OUTPUT ISOLATION

It was noted above that pins 10 and 11 are driven low during switching. Also during switching, some activity is seen on the PS/2 clock/data pins (20-23). It appears that the KVM switch temporarily disables the PS/2 output and sends reset signals to the mouse and keyboard as part of the switching process. The PS/2 protocol is bi-directional allowing the host to control features on the device, for example, setting indicators on a keyboard to show the state of caps lock, num lock, etc. The KVM switch needs to keep track of these features on each device (mouse and keyboard) per host/port, resetting them to their last known state for each host when switching between ports. To do this during switching, the KVM switch pulls pin 10 or 11 low to disable output to the corresponding output port then sends the appropriate signals on the PS/2 bus to the attached mouse or keyboard to reset their state. Pulling pin 10 or 11 low is necessary to prevent the attached host on the corresponding output port from seeing this signalling between the KVM switch and the mouse or keyboard.

In addition to this the KVM switch supports hotkey switching. This is triggered by pressing the 'Scroll Lock' key twice. At this point, the unit beeps (to indicate that it has entered its 'hotkey' state) and waits for additional keypresses for about 1 second. Any keypress during this time is interpreted by the KVM switch and not passed on to the connected computer. This functionality requires that the PS/2 output ports can be disabled while in this 'hotkey' state and this is achieved by pulling pin 10 or 11 low.

INCOMPLETE ANALYSIS

The analysis above has revealed a large proportion of the functionality of the KVM switch. However, there are still gaps. Several pins (12-19) were not seen to have any activity while running through a variety of KVM scenarios. If these pins were all high (normally a 'default' or 'inactive' state) then it could possibly be assumed that these pins are unused. However, the fact that some of these pins are low implies that there may be some purpose to them. Examination of traces on the PCB seem to show that only Pin 14 is connected, although its purpose is unclear.

As the analysis is incomplete, regenerating a working firmware from scratch is likely to be a challenge. Of the two options for subverting the KVM switch to implement a key-logger (see the 'Rationale' section above) the option of piggybacking a secondary microcontroller seems more tenable.

THE PS/2 INTERFACE

In order to access the keyboard data on the PS/2 interfaces in the KVM switch an understanding of the PS/2 protocol is required. A description of the PS/2 protocol can be found at the following link:

- http://www.computer-engineering.org/ps2protocol/

The PS/2 interface can be driven by either the host or the device, and the electrical characteristics of the interface mean that it should be possible to inject data onto the PS/2 bus even when both host and device are connected. This will be useful when attempting to piggyback a secondary microcontroller in the KVM switch.

© 2017 Cisco. All rights reserved.

IMPLEMENTING A KEYLOGGER

Given the analysis above, it should be possible to piggyback a secondary microcontroller on the PIC16C57C to implement a keylogger. Ideally this should be done using only signals on the pins of the PIC16C57C itself which would allow the same functionality to be implemented in replacement PIC firmware as an alternative.

Exfiltration should be via the existing interfaces on the KVM switch. The attacker would then only need in-situ access to the KVM switch in order to extract the data. One option is to implement an additional hotkey sequence in the secondary microcontroller. When this hotkey sequence is triggered, the secondary microcontroller would dump logged keypresses as keypress data to whatever system is connected to the KVM switch. This would allow an attacker to, for example, open a text editor on the target system and then press the hotkey sequence to dump the logged data into a text file.

ELECTRICAL CONSTRAINTS

There is a significant constraint when piggybacking on the PIC16C57C that any pin configured as an output on the PIC16C57C cannot be driven to a different state by the piggyback microcontroller. The PIC16C57C data sheet (section 7.6.1) says:

> *"A pin actively outputting a high or a low should not be driven from external devices at the same time in order to change the level on this pin ("wired-or", "wired-and"). The resulting high output currents may damage the chip."*

This means that an attempt to implement a keylogger and exfiltrate the logged data must use only pins on the PIC16C57C that are configured as inputs.

SHARING THE PS/2 BUS

One of the options for exfiltrating data from the keylogger is to send the data as a sequence of keypresses to the attached host computer. In order to do this, the KVM switch would have to take the role of the peripheral device (keyboard) on the PS/2 bus communicating with the host computer as a 'fake' keyboard. However, the PS/2 bus will also be connected to the real keyboard device at the same time so the question is whether exfiltration via the PS/2 bus can be achieved in this way.

The PS/2 bus is designed to be a point-to-point connection between a host computer and a peripheral device. It is a simple two-wire protocol (clock and data) and allows bi-directional communication. This is achieved by using an open-collector interface for both clock and data pins allowing either host computer or peripheral device to drive the state of either pin to low.

From a purely electrical standpoint, it is possible for a second peripheral device (the KVM switch) to drive the clock and data pins of the PS/2 interface. From the host computer both peripheral devices (real keyboard, and KVM switch acting as 'fake' keyboard) would be indistinguishable. However, there is a question of whether activity from the 'fake' keyboard may cause unexpected behaviour in the real keyboard.

In the PS/2 protocol, a majority of the communication is in the direction between the peripheral device and the host. The clock is always generated by the peripheral device meaning that the peripheral device is generating clock and data signals the majority of the time. If the host needs to send data to the peripheral device (for example, to set the state of LEDs, such as caps lock), the host first alerts the peripheral device that it wishes to send data by holding the clock line low for more than 100µs as part of a 'request to send' signal. Therefore, the concern is that activity on the PS/2 bus of the KVM switch acting as a 'fake' keyboard could be interpreted by the real keyboard as a 'request to send' from the host.

When sending data on the PS/2 bus, the

© 2017 Cisco. All rights reserved.

peripheral device generates a clock signal in the range 10-16.7kHz, therefore the clock cycle is 100µs at the slowest clock speed (10kHz). This means that the 'fake' keyboard will be pulling the clock low for a maximum of 50µs meaning that this should not be interpreted by the real keyboard as a 'request to send'. Therefore, sharing the PS/2 bus should be possible.

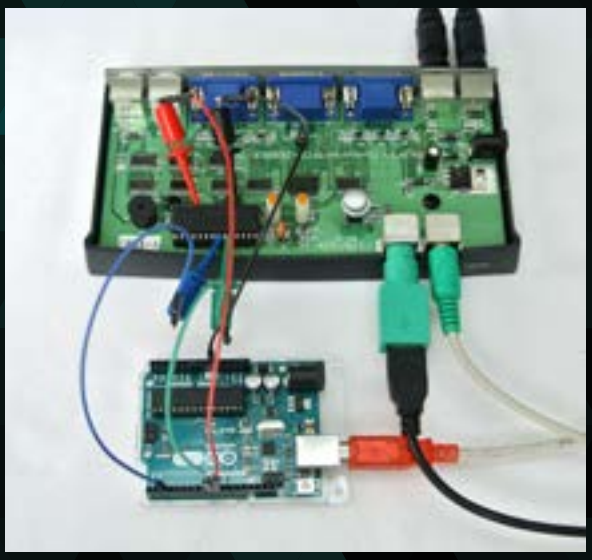
DETECTING THE HOTKEY SEQUENCE

Exfiltration of the logged data will be triggered by an additional hotkey sequence. The KVM switch implements hotkey sequences triggered by pressing the 'Scroll Lock' key twice within a certain timeframe. The PIC16C57C then pulls the appropriate 'output enable' pin low (pins 10 or 11 on the PIC16C57C) for about 1s to prevent further keypresses from being sent to the host, while it waits for follow-on keys in the hotkey sequence. For example, pressing the '1' or '2' keys will then cause a switch to output ports 1 or 2, respectively.

There are two options for implementing the hotkey sequence for exfiltration of logged data:

- Detect the triggering key sequence ('Scroll Lock' pressed twice) independently in the piggyback microcontroller.
- Detect the 'output enable' pins being pulled low, indicating that the hotkey sequence has been triggered.

The first option (independently detecting the hotkey trigger sequence) has a risk that the PIC16C57C and the piggyback microcontroller may not detect the hotkey trigger sequence in exactly the same way, for example, differences in timing. The second option based on the state of the 'output enable' pins will therefore be used, because it is a more reliable way to detect that the KVM switch has entered the hotkey trigger state.



Proof-of-Concept Keylogger Hardware

PROOF-OF-CONCEPT HARDWARE

In order to produce a proof-of-concept implementation of a keylogger an Arduino Uno development board was used. This has a range of I/O facilities and has a straight-forward 'C' based programming environment, providing easy access to I/O pins for prototyping.

The picture above shows the Arduino Uno board with the following piggyback connections onto the KVM switch PIC16C57C:

- **Black:** PIC16C57C pin 4 (Ground). Arduino ground.
- **Blue:** PIC16C57C pin 22 (PS/2 keyboard clock). Arduino digital I/O pin 2.
- **Green:** PIC16C57C pin 23 (PS/2 keyboard data). Arduino digital I/O pin 8.
- **Red:** PIC16C57C pin 11 ('Output enable' signal for KVM switch port 2). Arduino digital I/O pin 9.

For the purposes of the proof-of-concept implementation, only the Port 2 'output enable' line is being used.

© 2017 Cisco. All rights reserved.

PROOF-OF-CONCEPT SOFTWARE

CONTROL FLOW

The Arduino software to implement a keylogger will broadly have three states:

- While the 'output enable' is high (output enabled), log keypress data.
- When the 'output enable' is low (output disabled due to hotkey trigger sequence), detect the exfiltration hotkey sequence.
- If the exfiltration hotkey sequence has been detected and the 'output enable' transitions from low to high (becoming enabled again), exfiltrate the keypress data.

If the exfiltration hotkey sequence is detected, it is necessary to wait for the 'output enable' to go high again otherwise any PS/2 keypress data that is injected onto the PS/2 bus would not be seen by the host computer. For this proof-of-concept, the exfiltration hotkey sequence is set as two consecutive presses of the 'Q' key ('Q' for 'Query', and two presses so that it is not accidentally invoked by the legitimate user of the KVM switch).

KEYLOGGING

One advantage of using an Arduino development board is the large range of software libraries available. For this proof-of-concept the 'PS2Keyboard' library is used, which is available under the LGPL. This implements an interrupt-driven PS/2 keyboard reading library and automatically converts scan codes to ASCII, taking account of the shift key state to provide ASCII keypress data to the application. The Arduino Pin 2 is used for the PS/2 clock as this supports interrupts on the Arduino Uno board.

Due to the manner in which the Arduino board is connected to the KVM PIC16C57C, the same pins on the Arduino need to be used to both read PS/2 keyboard data during keylogging, and write PS/2 keyboard data during exfiltration. The 'PS2Keyboard' library only supports reading PS/2 data; a separate library is required to send data. In order to support this, a small addition to the 'PS2Keyboard' library had to be made in the addition of an end() method for the 'PS2Keyboard' library to release the interrupt used to read data. Without this addition, the 'PS2Keyboard' library would continue to read data during exfiltration.

During keylogging, keypress data received from this library is stored in a ring buffer ready for exfiltration. Note that this keypress data is ASCII rather than PS/2 scan codes so this needs to be taken into account during exfiltration.

EXFILTRATION

In order to exfiltrate logged key data over the PS/2 keyboard interface, a library for sending PS/2 keyboard data is required. A library needed to be written for this proof-of-concept as no such library was identified in the large selection of Arduino libraries freely available.

The PS/2 protocol uses a clock running in the 10-16.7kHz range and sends serial data with the following characteristics:

- One start bit (data held low).
- 8 data bits, LSB first.
- One parity bit (odd parity).
- One stop bit (data held high).

Data is read by the host on the falling edge of the clock (transition from high to low).

This data transmission format does not correspond with any of the built-in data transmission protocols in the Arduino meaning that a custom driver needed to be written. In order to implement the PS/2 protocol, a timer can be used that generates software interrupts at twice the rate of the 10-16.7kHz range required (in order to generate both rising and falling clock edges). The Arduino Uno supports 3 hardware timers that can generate software interrupts and Timer 2 was selected for this proof-of-concept; Timer 1 is used by some of the standard Arduino libraries, making it unsuitable.

© 2017 Cisco. All rights reserved.

A driver was written that manually sets the PS/2 clock and data pins on a 25kHz interrupt running off Timer 2 on the Arduino board. On each interrupt, the clock pin is driven successively high or low and the data pin is manipulated to generate the start bit, data bits, parity and stop bits as required.

When exfiltration is triggered by the hotkey sequence, logged data from the ring buffer needs to be exfiltrated. This is ASCII data so it cannot be sent directly as PS/2 scan codes. Instead, the data is exfiltrated as the hexadecimal representation of the ASCII of the logged keypresses. This can easily be converted back to the original keypress data once exfiltrated. For ease of conversion, the data was exfiltrated in a form compatible with the 'xxd' utility program available on Linux and other similar systems, similar to that shown below:

```
7373682074617267657473797374656d
0a726f6f740a50617373773072643132
330a
```

For each hex character output, the 'key depressed' keyboard scan code corresponding to that hex character (0-9 and A-F) is generated by the PS/2 driver, immediately followed by the 'key released' scan code for that same key. This continues until the entire ring buffer has been output.

The outcome of this exfiltration process is that the attached host computer sees a sequence of keypresses corresponding to the hex representation of the logged data. An attacker wishing to capture this data simply has to open a text editor on the target system, trigger the exfiltration, and watch as the exfiltrated data is 'typed' into the text editor.

SOURCE CODE

Source code for the proof-of-concept keylogger can be found here:

- https://labs.portcullis.co.uk/whitepapers/hacking-the-belkin-e-series-omniview-2-port-kvm-switch/

CONCLUSION

Hardware modification represents a genuine threat to organisations. Relatively simple hardware can, with the appropriate knowledge, be subverted to surreptitiously collect and ultimately exfiltrate data. Companies should remain abreast of the threat and consider conducting security appraisals of all devices deployed in sensitive areas.

Understanding 'normal' network traffic and remaining vigilant for unexpected and unusual network traffic, such as a new device suddenly connecting externally, can help organisations detect and block exfiltration over networks. However, in critical environments organisations need to identify and track hardware that is allowed to connect to critical systems including simple devices such as peripherals that are easy to overlook. The threat of physical tampering of devices means that in some environments physical examination of equipment is necessary to detect unauthorised modification.

© 2017 Cisco. All rights reserved.