Firmware validation

The logic to validate the firmware bundle begins at: `libupg_validate_firmware_mem`.

This is called from `libupg_upgrade_mem` that is subsequently called from the command `upgrade` in the `psbl` terminal.

This can be seen from the `upgrade` command handler:

```
int upgrade(int argc,char **argv)
{
  int iVar1;
  undefined4 *param2;

  if ((argc == 2 || argc == 4) && ((argc != 4 || (iVar1 = strcmp(argv[1],"-i"), iVar1 == 0)))) {
    *argv = "upgrade";
    argv[argc] = "/dev/ram";
    param2 = (undefined4 *)tftp(argc + 1,argv); // [1]
    if (0 < (int)param2) {
      iVar1 = libupg_upgrade_mem((astruct *)0xb4500000,param2); // [2]
      return iVar1;
    }
  }
  else {
    upgrade_usage();
  }
  return -1;
}
```

A couple of things to note from this snippet:

- [1] Shows the data is being obtained through `tftp` (wtf)
- We can see the address [2] `0xb4500000` being passed into `libupg_upgrade_mem`. This will be the location that the firmware is either downloaded to or where we will begin flashing.

libupg_validate_firmware_mem

The function starts by taking two arguments:

```
int libupg_validate_firmware_mem(byte *param1,uint param2)
[...]
```

- `param1` is the pointer into RAM discussed above (`0xb4500000`)
- `param2` is the result from `tftp` (Looks like it is being used as a data length)

The function begins by validating the firmware's header.

Header validation

The header validation can be found in function `validate_firmware_header`.

The function starts by taking a structure (we're going to call it `firmware_header_struct`).

The function contains two stages:

- Digest validation
- Randseq validation

Information regarding the "header format" has been split into a separate page. Please see Firmware format for more information. The rest of this section wiill use terminology sourced from it.

Digest validation

To calculate the digest you perform the following:

1. Zero out the Signature
2. Zero out the Digest
3. MD5 hash of the firmware header and module header table.

The size of the headers is calulcated with the formula below:

```
size = hdr->FirmwareHeaderSize + hdr->NumberOfModules * hdr->ModuleHeaderSize;
```

The process listed above can be seen in the following tidy Ghidra decompilation (Note: The `size` parameter is the one calulcated above)

```
int gen_fmhdr_digest(void *md5_struct_out, firmware_header_struct *fm_hdr, size_t size) {
  byte *pDigest;
  byte *pSignature;
  astruct_1 MD5Buffer [3];
  undefined lSignature [32];
  undefined lDigest [16];

  // Save signature and digest
  pSignature = fm_hdr->Signature;
  memcpy(lSignature,pSignature,0x20);

  pDigest = fm_hdr->Digest;
  memcpy(lDigest,pDigest,0x10);

  // Zero signature and digest
  memset(pSignature,0,0x20);
  memset(pDigest,0,0x10);

  // MD5 time
  MD5Init(MD5Buffer);
  MD5Update(MD5Buffer,fm_hdr,size);
  MD5Final(md5_struct_out,MD5Buffer);

  // Restore signature and digest
  memcpy(pSignature,lSignature,0x20);
  memcpy(pDigest,lDigest,0x10);
  return 0;
}
```

Randseq validation

To calculate the Randseq perform the following:

- Zero out the Signature, Digest and Ranseq elements of the header
- Perform the `nsdigest` on the same data as the MD5 digest above. `nsdigest` does the following:
  - For each 0x20 sized block in the firmware header and module header table:
    - For each byte:

        - Get the value from the byte array and add a special counter. Place this result of the addition into a temp buffer array
        - Increment counter by 0x19
    - Pass the 32 byte arrary into MD5Update
- Repeat until all bytes have been processed

This process can most clearly be seen with the following Python script:

```
import hashlib

SIGNATURE_OFFSET    = 16
DIGEST_OFFSET       = 48
RANDSEQ_OFFSET      = 64

def memset(data, start, size):
    temp_data_mutable = list(data)

    for i in range(size):
        temp_data_mutable[start+i] = 0

    return bytes(temp_data_mutable)

def nsdigest(data, size):
    data_index = 0
    addition_value = 0
    buf = [0] * 0x20

    md5 = hashlib.md5()

    while True:

        buffer_index = 0

        while True:
            buf[buffer_index] = (data[data_index] + addition_value) & 0xff

            addition_value += 19
            buffer_index += 1
            data_index += 1

            if buffer_index >= 0x20 or data_index >= size:
                break
```

```
        md5.update(bytes(buf))

        if data_index >= size:
            break

    return md5.digest().hex()

def main():

    with open("spa50x-30x-7-6-2g.bin", "rb") as f:
        data = f.read()

    # Calculates the size of the headers
    size = 0x80 + (0x4f * 0x40)

    data = memset(data, SIGNATURE_OFFSET, 32)
    data = memset(data, DIGEST_OFFSET, 16)
    data = memset(data, RANDSEQ_OFFSET, 16)

    x = nsdigest(data, size)
    print(x)

if __name__ == "__main__":
    main()
```

Revision #12
Created 30 March 2022 17:06:07 by Aidan
Updated 3 April 2022 17:52:07 by Aidan