SECCON CTF 2018 - re tctkToy writeup

sherlly / 2018-10-31 06:46:00 / 浏览数 2093 安全技术 CTF 顶(0) 踩(0)

前言

该题主要考察了TCL脚本的编写以及应用，TCL（Tool Command Language），是一种基于字符串的解释型命令语言，通常和Windows的GUI集成为Tk，易用C/C++/Java/Python扩展。

题目描述

The tctkToy was a fragile Windows application toy. Reverse and repair it in order to work well. SUPPORT: Recommend using Windows10 machine to run successfully.
P.S. File changed to this (28th 09:52JST(28th 00:52UTC)) file.zip_5bd5bdb6eaf308b509af1c466b8a76578b75cdd9
Hint : you can write a tcl file with just only "button", "exec", "cd", "wm", "canvas", "image" and "pack" command.

题目分析

题目启动时需传入1个参数：

```
if ( param1 != 1 )
{
  if ( param1_ != 2 )
  {
    v78 = MessageBoxA(0, "Seek the start-up sequence.", "ERROR", 0);
    sub_F2135C(v80, v79, &v112 == &v112, v78, a1);
    exit(1);
  }
  v35 = lstrcpyA(&String1, "help message!!\n");
  sub_F2135C(v37, v36, &v113 == &v113, (int)v35, a1);
  v112 = 0;
  v111 = &NumberOfCharsWritten;
  v38 = lstrlenA(&String1);
  v41 = sub_F2135C(v40, v39, &v111 == &v111, v38, a1);
  v42 = WriteConsoleA(hConsoleOutput, &String1, v41, v111, v112);
  sub_F2135C(v44, v43, &v113 == &v113, v42, a1);
  v45 = lstrcpyA(
          &String1,
          "This tctkToy was old fragile and small Windows application toy.. One day, my baby broke it and I forgot how "
          "to the repair.\n");
  sub_F2135C(v47, v46, &v113 == &v113, (int)v45, a1);
  v112 = 0;
  v111 = &NumberOfCharsWritten;
  v48 = lstrlenA(&String1);
  v51 = sub_F2135C(v50, v49, &v111 == &v111, v48, a1);
  v52 = WriteConsoleA(hConsoleOutput, &String1, v51, v111, v112);
  sub_F2135C(v54, v53, &v113 == &v113, v52, a1);
```

当参数值为2时控制台打印帮助信息如下：

```
tctkToy.exe 2
2   迦elp message!!
This tctkToy was old fragile and small Windows application toy.. One day, my baby broke it and I forgot how to the repair.
Please reconstruct my tctkROBO with reverse engineering. It eats the procedure script and can be built.
This toy somehow cares about its task resources.
```

当参数值为1时控制台打印信息`build & check mode`，初始化显示的窗体：

```
v25 = lstrcpyA(&String1, "build & check mode\n");
sub_F2135C(v27, v26, &v113 == &v113, (int)v25, a1);
v112 = 0;
v111 = &NumberOfCharsWritten;
v28 = lstrlenA(&String1);
v31 = sub_F2135C(v30, v29, &v111 == &v111, v28, a1);
v32 = WriteConsoleA(hConsoleOutput, &String1, v31, v111, v112);
sub_F2135C(v34, v33, &v113 == &v113, v32, a1);
v81 = LocalFree(hMem);
sub_F2135C(v83, v82, &v111 == &v111, (int)v81, a1);
v84 = LoadStringW(hInstance, 0x67u, (LPWSTR)&WindowName, 100);
sub_F2135C(v86, v85, &v111 == &v111, v84, a1);
v87 = LoadStringW(hInstance, 0x6Du, (LPWSTR)&ClassName, 100);
sub_F2135C(v89, v88, &v111 == &v111, v87, a1);
j_init_window(a1, hInstance);
v90 = sub_F210FA(a1, hInstance, nCmdShow);
```

```
sub_F21343(&unk_F3004D);
v20.cbSize = 48;
v20.style = 3;
v20.lpfnWndProc = (WNDPROC)j_handle;
v20.cbClsExtra = 0;
v20.cbWndExtra = 0;
v20.hInstance = hInstance;
v2 = LoadIconA(hInstance, (LPCSTR)0x6B);
v20.hIcon = (HICON)sub_F2135C(v4, v3, &v19 == &v19, (int)v2, a1);
v5 = LoadCursorA(0, (LPCSTR)0x7F00);
v20.hCursor = (HCURSOR)sub_F2135C(v7, v6, &v19 == &v19, (int)v5, a1);
v20.hbrBackground = (HBRUSH)6;
v20.lpszMenuName = (LPCWSTR)109;
v20.lpszClassName = &ClassName;
v8 = LoadIconA(v20.hInstance, (LPCSTR)0x6C);
v20.hIconSm = (HICON)sub_F2135C(v10, v9, &v19 == &v19, (int)v8, a1);
LOWORD(v11) = RegisterClassExW(&v20);
v14 = (char *)sub_F2135C(v13, v12, &v19 == &v19, v11, a1);
v16 = v15;
sub_F21398(v14);
return sub_F2135C((unsigned int)&savedregs ^ v21, v16, 1, v17, a1);
}
```

进入和窗体绑定的回调函数handle，可以看到处理消息中，当消息值为1时，开启了拖拽文件的开关，并且加载了资源的位图文件：

```
case 1u:
  DragAcceptFiles(hWnd, 1);
  sub_F2135C(v7, v6, &v77 == &v77, v5, a1);
  v93 = lParam;
  v8 = LoadBitmapA(*(HINSTANCE *)(lParam + 4), (LPCSTR)0x65);
  ho = (HANDLE)sub_F2135C(v10, v9, &v77 == &v77, (int)v8, a1);
  break;
```

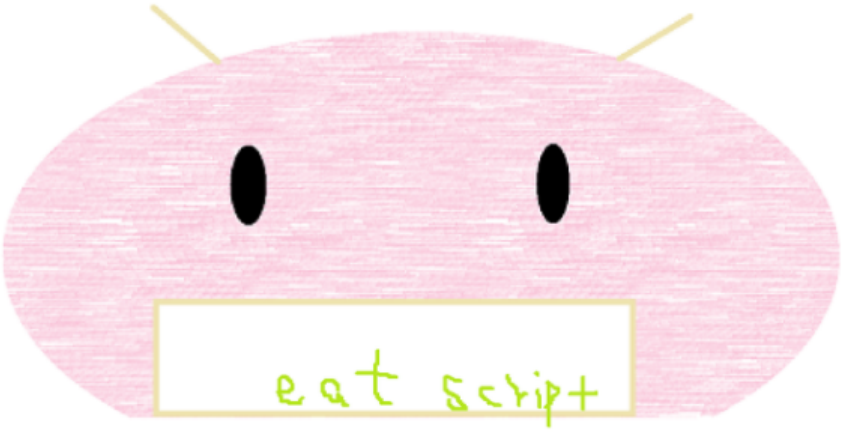
接下来创建新线程对拖拽入窗体的文件进行处理：

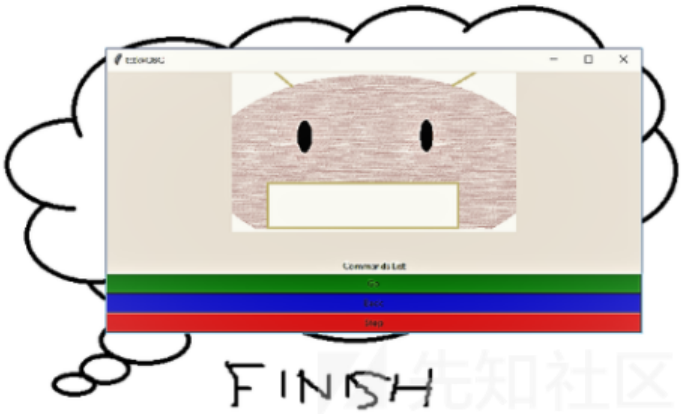
```
  v12 = SetForegroundWindow(hWnd);
  sub_F2135C(v14, v13, &v77 == &v77, v12, a1);
  v15 = DragQueryFileA(hDrop, 0, tcl_script, 0x104u);
  sub_F2135C(v17, v16, &v77 == &v77, v15, a1);
  v18 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)j_evalTcl, tcl_script, 0, &ThreadId);
  v92 = sub_F2135C(v20, v19, &v77 == &v77, (int)v18, a1);
  Sleep(0x3E8u);
  sub_F2135C(v23, v22, &v77 == &v77, v21, a1);
  DragFinish(hDrop);
  sub_F2135C(v26, v25, &v77 == &v77, v24, a1);
```

可以看到处理的函数中将文件作为TCL脚本进行执行：

```
sub_F21343(&unk_F3004D);
v2 = Tcl_CreateInterp();
v5 = sub_F2135C(v4, v3, &v35 == &v35, v2, v1);
v37 = v5;
v6 = Tcl_Init(v5);
sub_F2135C(v8, v7, &v35 == &v35, v6, v1);
v9 = Tk_Init(v37);
sub_F2135C(v11, v10, &v35 == &v35, v9, v1);
v12 = Tcl_EvalFile(v37, a1);
if ( sub_F2135C(v14, v13, &v35 == &v35, v12, v1) )
{
  v34 = 0;
  v15 = Tcl_GetStringResult(v37);
  v18 = (const CHAR *)sub_F2135C(v17, v16, &v33 == (int *)&v34, v15, v1);
  v19 = MessageBoxA(0, v18, v34, v35);
  sub_F2135C(v21, v20, &v35 == (UINT *)&v36, v19, v1);
  v22 = MessageBoxA(0, "It cannot eat this completely.", "Oops!", 0);
  sub_F2135C(v24, v23, &v36 == &v36, v22, v1);
  exit(1);
}
v25 = Tk_MainLoop();
sub_F2135C(v27, v26, &v35 == &v35, v25, v1);
v28 = Tcl_Finalize();
sub_F2135C(v30, v29, &v35 == &v35, v28, v1);
sub_F2135C(v32, v31, 1, 0, v1);
```

结合加载的位图提供的信息，可以知道程序需要在执行拖拽入的TCL脚本文件后出现修复后的窗口样式：





解题过程

在执行完TCL脚本后，进行两处check，通过则打印flag：

```
    if ( j_checktcl() )
    {
      flag = (char *)j_read_tcl(tcl_script);
      j_memset(&Dst, 0, 0x6Eu);
      v27 = strncmp(flag, "fail", 4u);
      if ( sub_F2135C(v29, v28, &v77 == &v77, v27, a1) )
      {
        sub_F21451(&Dst, 110, "congraturation!! flag is SECCON{%s}", flag);
        v30 = MessageBoxA(0, &Dst, "Complete!", 0);
      }
      else
      {
        v30 = MessageBoxA(
                hWnd,
                "review the order following FINISH view. 'pack' must be used once for each component",
                "If you cannot pass the flag??",
                0);
      }
      sub_F2135C(v32, v31, &v77 == &v77, v30, a1);
    }
```

第一处check主要针对TCL脚本执行后带来的变化，分为四处：

（1）检测了当前工作目录是否为C:\tctkToy：

```
v4 = GetCurrentDirectoryA(0x105u, &Buffer);
sub_F2135C(v6, v5, &v57 == &v57, v4, a1);
if ( j_strcmp(&Buffer, ::Buffer) )
{
  v7 = strncmp(&Buffer, "C:\\tctkToy", 0xBu);
  if ( !sub_F2135C(v9, v8, &v57 == &v57, v7, a1) )
    *check_flag = 1;
}
```

（2）遍历进程列表是否存在任务管理器Taskmgr.exe并进行关闭：

```
hSnapshot = j_CreateToolhelp32Snapshot(2u, 0);
if ( j_Process32First(hSnapshot, &pe) == 1 )
{
  while ( j_Process32Next(hSnapshot, &pe) == 1 )
  {
    v10 = stricmp(pe.szExeFile, "Taskmgr.exe");
    if ( !sub_F2135C(v12, v11, &v57 == &v57, v10, a1) )
    {
      v13 = OpenProcess(0x411u, 0, pe.th32ProcessID);
      hProcess = (HANDLE)sub_F2135C(v15, v14, &v57 == &v57, (int)v13, a1);
      check_flag[1] = 1;
      v16 = TerminateProcess(hProcess, 1u);
      sub_F2135C(v18, v17, &v57 == &v57, v16, a1);
      v19 = CloseHandle(hProcess);
      sub_F2135C(v21, v20, &v57 == &v57, v19, a1);
    }
  }
}
v22 = CloseHandle(hSnapshot);
```

（3）检查是否存在标题为tctkROBO的窗口：

```
v28 = FindWindowA(0, "tctkROBO");
hWnd = (HWND)sub_F2135C(v30, v29, &v57 == &v57, (int)v28, a1);
if ( hWnd )
{
  check_flag[2] = 1;
  String = 0;
  j_memset(&Dst, 0, 0xFFu);
```

（4）检查窗体TkChild控件和Button控件的数量需满足count(TkChild)*10+count(Button)==13：

遍历控件并计数：

```
sub_F21343(&unk_F3004D);
v14 = (_DWORD *)a3;
v3 = GetWindowTextA(hWnd, &String, 1024);
sub_F2135C(v5, v4, &v13 == &v13, v3, a1);
v6 = GetClassNameA(hWnd, &ClassName, 1024);
sub_F2135C(v8, v7, &v13 == &v13, v6, a1);
if ( *v14 )
{
  if ( !j_strcmp(&String, &Str2) && !j_strcmp(&ClassName, "Button") )
  {
    v9 = v14;
    ++*v14;
  }
}
else if ( !j_strcmp(&String, &Str2) && !j_strcmp(&ClassName, "TkChild") )
{
  v9 = v14;
  *v14 += 10;
}
v10 = (int)v9;
sub_F21398((char *)1);
return sub_F2135C((unsigned int)&savedregs ^ v17, v10, 1, v11, a1);
```

检查数量是否为13：

```
v38 = EnumChildWindows(hWnd, EnumFunc, (LPARAM)&lParam);
sub_F2135C(v40, v39, &v57 == &v57, v38, a1);
if ( lParam == 13 )
  check_flag[3] = 1;
```

第二处check针对TCL脚本本身内容进行：

（1）对每行的指令语句取头两个字符进行连接，其中头两个字符为.时不选取，且读取的总长度为24，即实际读取的指令语句应该是12条：

```
v2 = CreateFileA(tcl_script, 0x80000000, 1u, 0, 3u, 0x80u, 0);
v5 = (void *)sub_F2135C(v4, v3, &v25 == &v25, (int)v2, a1);
hFile = v5;
v6 = ReadFile(v5, &Buffer, 0x400u, &NumberOfBytesRead, 0);
sub_F2135C(v8, v7, &v25 == &v25, v6, a1);
v9 = strncpy_s(Dst, 3u, &Buffer, 0xFFFFFFFF); // 选取头两个字符
sub_F2135C(v11, v10, &v25 == &v25, v9, a1);
if ( dword_F2D558 > *(_DWORD *)(*(_DWORD *)(__readfsdword(0x2Cu) + 4 * TlsIndex) + 260) )
{
  sub_F21122((int)&dword_F2D558);
  if ( dword_F2D558 == -1 )
  {
    Str = &Buffer;
    sub_F21221((int)&dword_F2D558);
  }
}
while ( 1 )
{
  Str = (char *)findch(Str, '\n');
  if ( !Str )
    break;
  v12 = strncpy_s(Src, 3u, Str + 1, 0xFFFFFFFF);// 选取头两个字符
  sub_F2135C(v14, v13, &v25 == &v25, v12, a1);
  if ( findch(Src, '\n') )
    break;
  if ( !findch(Src, '.') )                    // 头两个字符包含.时不选取
  {
    v15 = strcat_s(Dst, 25u, Src);            // 最多连接24个字符
    sub_F2135C(v17, v16, &v25 == &v25, v15, a1);
  }
  ++Str;
}
```


（2）对连接后的指令内容进行SHA256哈希，将头20个字符和a683618184fc18105b71比较，相等则通过：

```
v7 = CryptCreateHash(hProv, 0x800Cu, 0, 0, &hHash);// sha256
if ( sub_F2135C(v9, v8, &v28 == &v28, v7, a1) )
{
  Dst = 0;
  v34 = 0;
  v35 = 0;
  v36 = 0;
  v37 = 0;
  v38 = 0;
  v39 = 0;
  j_memcpy(&Dst, Src, 25u);
  v10 = CryptHashData(hHash, (const BYTE *)&Dst, 24u, 0);
  if ( sub_F2135C(v12, v11, &v28 == &v28, v10, a1) )
  {
    Size = 32;
    v13 = malloc(0x20u);
    hash_res = (BYTE *)sub_F2135C(v15, v14, &v28 == &v28, (int)v13, a1);
    if ( hash_res )
    {
      v16 = CryptGetHashParam(hHash, 2u, hash_res, &Size, 0);
      if ( sub_F2135C(v18, v17, &v28 == &v28, v16, a1) )
      {
        for ( i = 0; i < Size; ++i )
        {
          v29 = 0;
          v30 = 0;
          sub_F21451(&v29, 3, "%02x", hash_res[i]);// hexlify
          v19 = strcat_s(hex_hash, 65u, &v29);
          sub_F2135C(v21, v20, &v28 == &v28, v19, a1);
        }
        v22 = strncmp(hex_hash, "a683618184fc18105b71", 20u);
        if ( sub_F2135C(v24, v23, &v28 == &v28, v22, a1) )
          v6 = "fail";
        else
          v6 = hex_hash;
```

TCL脚本

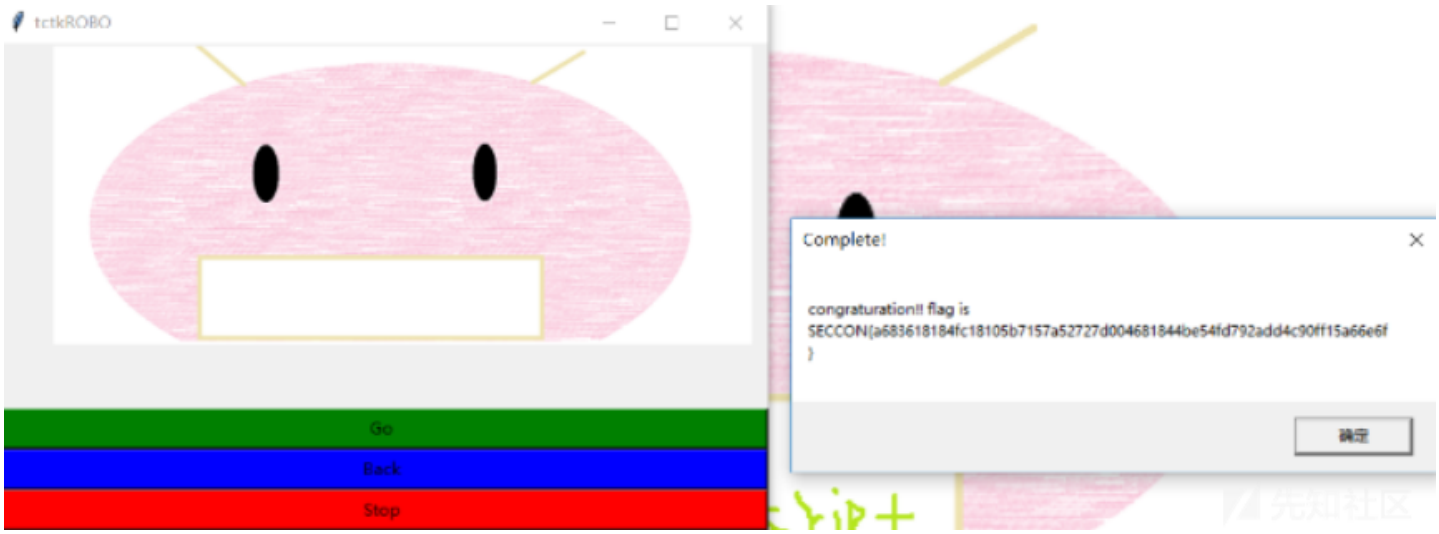
```
cd "C:\\tctkToy"
exec "Taskmgr.exe" "/c"
wm title . "tctkROBO"
canvas .c
image create photo img -file face.png
.c create image 300 110 -image img
pack .c -expand yes -fill both -side top
button .go -text "Go" -command exit -background green -width 80
pack .go
button .back -text "Back" -command exit -background blue -width 80
pack .back
button .stop -text "Stop" -command exit -background red -width 80
pack .stop -side bottom
```

有个坑点就是TCL在执行启动Taskmgr.exe时会报invalid argument错误，原因可能是启动的权限不够（在管理员模式下可以启动），这里采用的解决方法是在C:\tctkToy工作目录下放一个cmd.exe（重命名为Taskmgr.exe），在

解题脚本

```
import hashlib
# "button", "exec", "cd", "wm", "canvas", "image" and "pack"
key="cdexwm"
key+="caimpa"
key+="bupabupabupa"
assert len(key)==24
m=hashlib.sha256()
m.update(key)
sha = m.hexdigest()
if sha[:20]=="a683618184fc18105b71":
   print "SECCON{%s}"%sha
# SECCON{a683618184fc18105b7157a52727d004681844be54fd792add4c90ff15a66e6f}
```

结果



参考资料

http://zetcode.com/gui/tcltktutorial/drawing/
https://www.tcl.tk/man/tcl/TkCmd/button.htm
https://wiki.tcl-lang.org/page/Simple+Canvas+Demo

file.zip (2.37 MB) 下载附件

上一篇：渗透测试之Jarbas和Foura... 下一篇：如何绕过AMSI并执行任意恶意Po...

1. 0 条回复
   - 动动手指，沙发就是你的了！