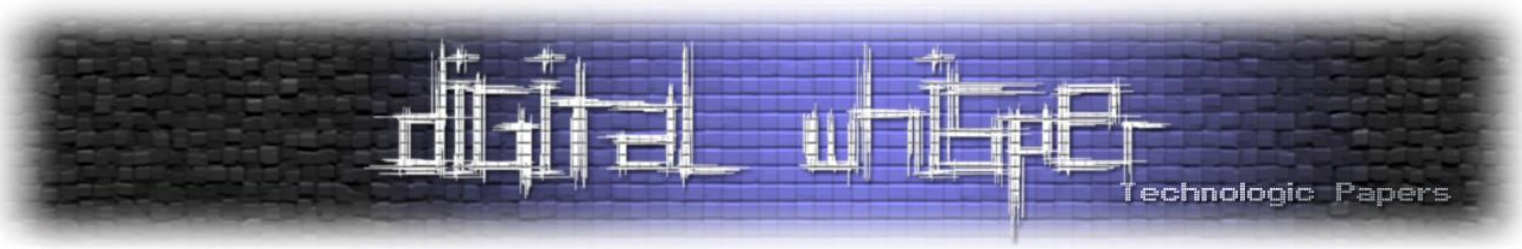


Windows Notification Facility

מאת טל בלום

הקדמה

החל מ-Windows 8, התווסף לקרנל מנגנון חדש בשם Windows Notification Facility, או בקיצור - WNF. WNF הוא מנגנון פנימי של הקרנל בשיטת Subscriber\Publisher להעברת מידע והתראות על אירועים במערכת ההפעלה בין רכיבים שונים למטרות סנכרון. המידע נגיש גם מ-Kernel Mode וגם מ-Mode User, וגם בין תהליכים שונים. המנגנון הלך וגדל בין מערכות ההפעלה והיום יש לו שימושים רבים בפעילות הפנימית של מערכת ההפעלה.

ב-Black Hat האחרון (2018) Alex Ionescu, מכותבי הספר Windows Internals, ביחד עם Gabrielle Viala, הציגו את ה-WNF וסוגיות אבטחה שהוא מעלה [1]. עד ההרצאה כמעט ולא היה בכלל מידע באינטרנט על המנגנון הזה, שהוא לחלוטין Undocumented.

במאמר זה אסביר מה הוא אותו WNF, איך המנגנון הזה עובד, את המבנה שלו ומה השימושים שלו במערכת ההפעלה. בחלק השני של המאמר אני אציג רעיונות כיצד ניתן לנצל אותו לרעה כדי לעקוף כלי EDR, ואציג ווקטורי תקיפה ואפשרויות מעניינות אחרות שהוא מספק ב-Windows.

מה זה Windows Notification Facility

WNF פועל בשיטה של Publisher ו-Subscriber. כלומר, רכיב (יכול להיות או דרייבר או תהליך ב-User Space) יכול לעשות Subscribe לאירוע מסוים (נקרא StateName), על מנת לקבל התראה בכל פעם שרכיב אחר עשה "Publish" אל אותו State Name. כלומר, ביצע שינוי במידע שעליו (נקרא StateData).

לדוגמא, כשמערכת ההפעלה עושה Boot, ה-HD קודם כל עולה כ-Read-Only כדי ש-AutoChk יבדוק את תקינות ה-HD. ברגע שהוא מסיים את הבדיקה, הדיסק עובר Mount שוב עם הרשאות של RW ונשלח WNF Event שמיידע את שאר הרכיבים שעשו Subscribe לאותו State Name שהדיסק תקין וניתן לרשום כעת אל ה-HD.

נכון לגרסא החדשה ביותר של Windows10 (1809) קיימים כ-4,000 אירועי WNF במערכת ההפעלה. WNF משמש כדי לתקשר בין תהליכים, בין Sessions, ובין User Mode ל-Kernel Mode. המגבלה היחידה שלו היא שכל State Name יכול להכיל State Data רק עד ל-4KB.

הרצה של תהליך גורמת להתראת WNF, סגירת תהליך גורם להתראת WNF, חיבור של רכיב שמע גורם להתראת WNF, חיבור לרשת Wi-Fi גורם להתראת WNF, כשהמערכת זיהתה שהמשתמש לא פעיל (המחשב לא קיבל קלט בפרק זמן מסוים) הדבר גורם להתראת WNF, ועוד רבים.

ה-WNF הוא לא שכבה מעל ETW או ALPC, או כל מנגנון אחר של הקרנל, ולא קשור אליהם. אלא מנגנון נפרד, בעל שימושים פנימיים בלבד של מערכת ההפעלה, ונגיש רק דרך כמה פונקציות לא מתועדות.

העובדה שהוא עדיין יחסית לא מוכר גורמת למנגנון להיות מעניין ביותר. כלי EDR כמו Sysmon לא מכירים אותו ואין להם שום דרך לזהות שימוש בו. תוקפים יכולים לנצל את המנגנון להסתרת הפעילות הזדונית שלהם וככה לעקוף מערכות הגנה. על כך נדבר בהרחבה בהמשך.

WNF הוא מנגנון אשר קיים גם בפלטפורמות אחרות של Microsoft, כמו Windows Phone או XBox. כך שיש גם התראות WNF על יצירת שיחת טלפון ועוד התראות רבות אחרות שרלוונטיות רק לגרסאת המובייל או ל-XBox, אך עדיין קיימות בכל הפלטפורמות.

מעבר לאפשרות שלנו לקריאה וכתיבה של ה-State Data ולהירשם ל-State Names שכבר קיימים במערכת ההפעלה, יש API גם ליצירת State Names חדשים.

WNF State Names מעניינים / רגישים

מתוך המוני ה-State Names הקיימים במערכת ההפעלה (שרובם לא מעניינים אותנו כאנשי אבטחת מידע), יש כמה שכן יכולים לספק לנו מידע מעניין ואפילו לשנות את פעילות המכונה:

- WNF_WIFI_CONNECTION_STATUS - מודיע בכל פעם שהמחשב מתחבר ל-WI-FI
- WNF_AUD_CAPTURE\RENDER - מודיע לנו כל פעם שתהליך משמיע\מקליט סאונד. כולל שם התהליך שהפעיל את זה.
- WNF_SEB_USER_PRESENT \ WNF_SEB_USER_PRESENCE_CHANGED - מודיע כאשר משתמש לא היה פעיל X זמן, לפי ההגדרה של Windows.
- WNF_SHEL_(DESKTOP) APPLICATION_(STARTED \ TERMINATED) - נותן לך את השם של כל תכנית שהתחילה\נסגרה.
- WNF_EDGE_LAST_NAVIGATED_HOST - ה-URL האחרון שהמשתמש כתב לתוך הדפדפן Edge.
- WNF_SYS_SHUTDOWN_IN_PROGRESS - נותן התראה כשהמחשב בתהליך כיבוי.

- WNF_FSRL_OPLOCK_BREAK - מקבל רשימה של PID-ים וסוגר אותם!

כדי להראות עוד כמה מהשימושים של WNF במערכת ההפעלה, שווה להסתכל על פרוייקט מגניב בגיטהאב בשם Mach2:

https://github.com/riverar/mach2/

ב-Windows יש מה שנקרא "Insider Preview Features". כש-Windows רוצים לבדוק פיצ'ר חדש מסוים, הם מחילים אותו רק על חלק מהמשתמשים כדי לראות את התגובה שלהם ולדעת מה אהוד על ידי המשתמשים ומה לא. לדוגמא הפיצ'ר Dark Mode, שמאפשר לשנות את העיצוב של האפליקציות של Windows מלבנות ומוארות לשחורות ואפלות.

יש אלפי פיצ'רים כאלה, שהסוויץ' האם הם יהיו מופעלים או לא נמצא כמו שניחשתם בתוך WNF. הפרויקט הזה משנה את ערכי ה-State Names הרלוונטים, ואתה יכול בעצם להחליט בעצמך איזה פיצ'רים יהיו לך או לא יהיו לך.

ככה בעצם אפשר להרוס ל-Microsoft את כל הבדיקות שלהם, כי כאמור ה-WNF הוא מנגנון לא מתועד שהמשתמשים לא אמורים לשחק בו.

מבנה ה-State Name

ישנם 4 סוגים של State Names:

- Well-Known - State Names שמגיעים מוגדרים עם מערכת ההפעלה, לא ניתן להגדיר חדשים מהסוג הזה. המידע שבהם הוא Persistent, כלומר נשמר גם לאחר שהמחשב נכבה. נמצאים ברג'יסטרי תחת הנתיב:

```
HKLM\SYSTEM\CurrentControlSet\Control\Notifications
```

- Permanent - המידע בהם יכול להיות Persistent ויכול להיות שלא. תלוי ב-flag IsPermanent מצאים גם כן ברג'יסטרי תחת הנתיב:

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Notifications
```

- Persistent - הם Persistent בהקשר שהם נשמרים גם אחרי שהתהליך שרשם אותם נסגר, אך לא נשמרים לאחר שהמחשב נכבה. נמצאים ברג'יסטרי תחת הנתיב:

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\VolatileNotifications
```

- Temporary - פחות נפוצים. כשהתהליך נסגר, ה-State Name הזמני נעלם גם הוא. אין לו רשומה ברג'יסטרי.

כדי ליצור Persistent או Permanent Names צריך הרשאות Administrator.

ה-State Names עצמם מוגדרים במערכת ההפעלה עם ID ייחודי. עבודת Reverse Engineering, שהתחיל אותה redplaint בבלוג שלו [2], הוציאה מתוך perf_nt_c.dll את רשימת ה-IDים. שלמרות שהם Undocumented, בתוך ה-dll יש מחרוזת שמציינת את השם שלהם והסבר קצר. רשימה יותר מלאה שמקשרת בין שם ה-State Name ל-ID שלו, נמצאת כאן: [3].

ה-ID של כל State Name הוא בעצם ערך שמורכב מהשדות: Version, Lifetime, Scope, Data permanence flag ו-Unique id. שעליהם עשו XOR עם ערך קבוע: 0x41C64E6DA3BC007:

```
typedef struct _WNF_STATE_NAME_INTERNAL
{
    ULONG64 Version : 4;
    ULONG64 Lifetime : 2;
    ULONG64 DataScope : 4;
    ULONG64 IsPermanent : 1;
    ULONG64 Unique : 53;
} WNF_STATE_NAME_INTERNAL, *PWNF_STATE_NAME_INTERNAL;
```

[https://blog.quarkslab.com/playing-with-the-windows-notification-facility-wnf.html]

Lifetime - מתייחס לארבעת הסוגים של State Names (WellKnown, Permanent, Persistent, Temporary):

```
typedef enum _WNF_STATE_NAME_LIFETIME
{
    WnfWellKnownStateName = 0x0;
    WnfPermanentStateName = 0x1;
    WnfPersistentStateName = 0x2;
    WnfTemporaryStateName = 0x3;
} WNF_STATE_NAME_LIFETIME;
```

State names - Scope יכולים להיות גלובאליים, או ספציפיים ל-Session, או ל-User או ל-Process.

```
typedef enum _WNF_DATA_SCOPE
{
  WnfDataScopeSystem = 0x0,
  WnfDataScopeSession = 0x1,
  WnfDataScopeUser = 0x2,
  WnfDataScopeProcess = 0x3,
  WnfDataScopeMachine = 0x4,
} WNF_DATA_SCOPE;
```

ה-IsPermanent Flag רלוונטי ל-Persistent State Names אשר יכולים להיות Persistent לאחר כיבוי המחשב ויכולים שלא.

כדי לפענח את המידע של השדות האלה, יש קודם כל לבצע XOR של ערך ה-State name עם הערך הקבוע (שלא ברור מה המשמעות שלו):

```
WNF_XOR_KEY = 0x41C64E6DA3BC0074;
```

כך לדוגמא, יש את ה-State Name שנקרא WNF_SYS_SHUTDOWN_IN_PROGRESS אשר מיידע כאשר המחשב נמצא בתהליך כיבוי. וה-ID המייצג אותו במערכת ההפעלה הוא: 0x4195173ea3bc0875.

נעשה את הפעולות הבאות כדי לפענח:

```
#include <iostream>
#define WNF_XOR_KEY 0x41C64E6DA3BC0074
using namespace std;

int main() {
long int StateName = 0x4195173ea3bc0875;
long int ClearStateName = StateName ^ WNF_XOR_KEY;
cout<<"Clear State Name: "<<hex<<ClearStateName<<endl;
long int Version = ClearStateName & 0xf;
cout<<"Version: "<<hex<<Version<<endl;
long int LifeTime = (ClearStateName >> 4) & 0x3;
cout<<"LifeTime: "<<hex<<LifeTime<<endl;
long int DataScope = (ClearStateName >> 6) & 0xf;
cout<<"Scope: "<<hex<<DataScope<<endl;
long int IsPermanent = (ClearStateName >> 0xa) & 0x1;
cout<<"IsPermanent: "<<hex<<IsPermanent<<endl;
long int Unique = ClearStateName >> 0xb;
cout<<"Unique ID: "<<hex<<Unique<<endl;
}
```

כך שמאפייניו של WNF_SYS_SHUTDOWN_IN_PROGRESS הם:

```
Clear State Name: 53595300000801
Version: 1
LifeTime: 0   Well Known
Scope: 0    System
IsPermanent: 0   Nope
Unique ID: a6b2a600001
```

WNFUN

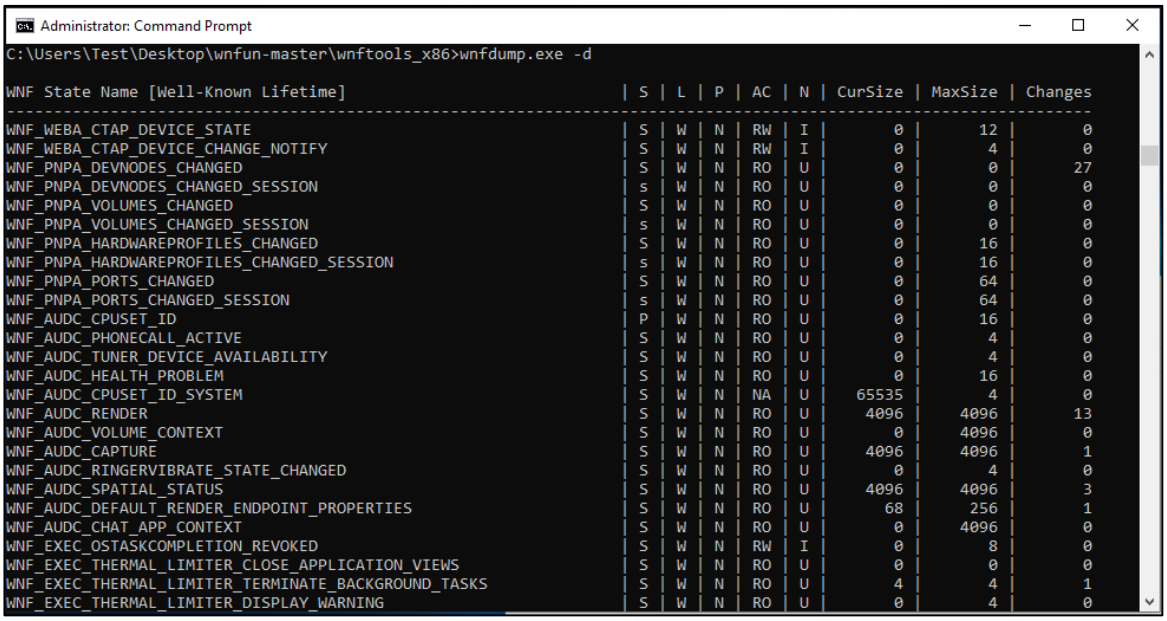
בהרצאה של Alex ו-Gabrielle, הוצגו כלים שנכתבו על ידם לשימוש ב-WNF ולאחר מכן גם הועלו ל-Github:

https://github.com/ionescu007/wnfun/

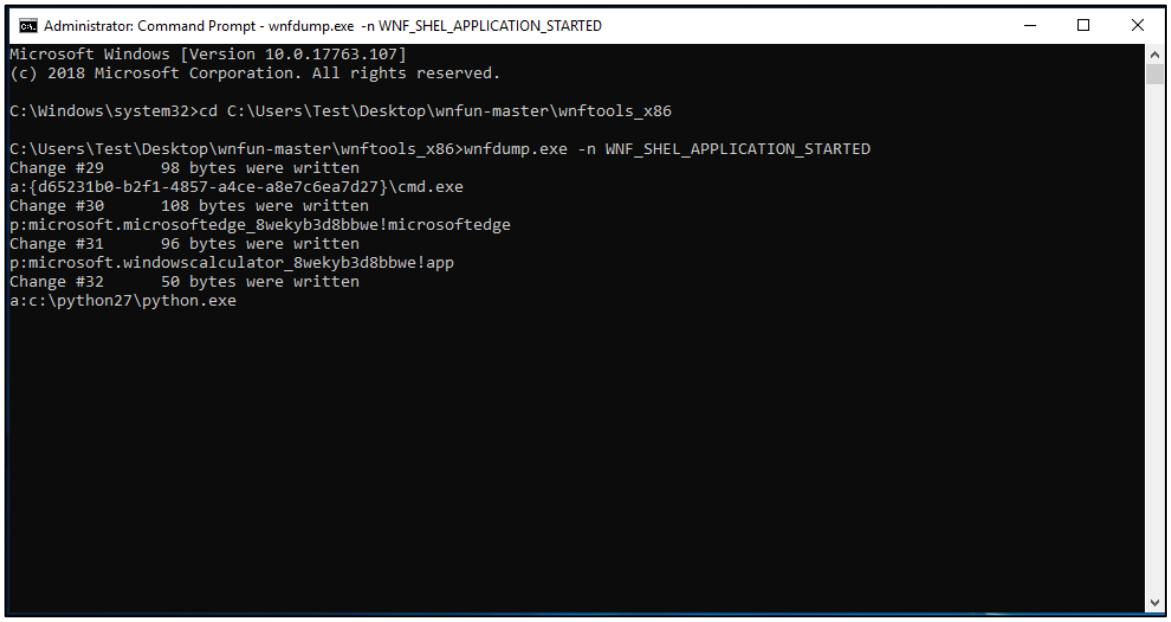
הכלים כוללים את WNFDump - היום מדובר בכלי עם הכי הרבה יכולות, והכי נח לשימוש ב-WNF. הוא מאפשר לך לחפש State Names מעניינים (גם דרך הרג'יסטרי וגם דרך ברוט-פורס של ה-ID במידה ומדובר ב-Temporary State Name).

הוא מאפשר לך לקרוא את המידע השמור בהם, ונותן לך לכתוב בעצמך אליהם ולראות איך זה משנה את פעילות מערכת ההפעלה. בנוסף הוא מאפשר לך להירשם ל-State Name מסוים ומציג כל פעם שקרה בו שינוי. כדאי לציין שאם מוחקים את כל המידע שבכל ה-State Names, המחשב נהרס - ה-Explorer קורס בלי דרך להחזיר אותו, ואתם בעצם לא יכול יותר לעשות כלום. וגם Reboot לא יתקן אותו.

דוגמא לפלט של הכלי להוצאת כל ה-State Names דרך הרג'יסטרי (כמובן שהרשימה ממשיכה עוד הרבה):



והנה דוגמא להרשמה ל-WNF_SHEL_APPLICATION_STARTED, ה-State Name הזה מתעדכן כל פעם שנפתחת תוכנה במחשב (עם GUI). פתחתי כמה תוכנות לבדיקה וראיתי איך זה מתעדכן:



בנוסף הם סיפקו כלים כדי ליישם Client ו-Server - מאפשר לך ליצור State Name משלך ולהירשם אליו. הכלים פורסם ללא ה-Source code.

יש בפרוייקט ב-Github גם סקריפטים של Python, שמיישמים את אותו דבר. אך עם תיעוד טוב של המבנים בזיכרון, הפונקציות הלא מתועדות שבהם נעשה שימוש וכוללים רשימה מועילה של ה-ID-ים של ה-Well Known Name States.

WNF API (איך WNFUN עובד)

אף אחת מהפונקציות שמתעסקות עם WNF לא מתועדת, אך יש פרוייקטים באינטרנט שחקרו את הפונקציות הלא מתועדות ב-Windows ופרסמו אותן [5]. כדי ליצור State Name חדש משלנו ניתן להשתמש ב:

```
NTSTATUS ZwCreateWnfStateName (
_Out_    PWNF_STATE_NAME      StateName,
_In_     WNF_STATE_NAME_LIFETIME      NameLifeTime,
_In_     WNF_DATA_SCOPE  DataScope,
_In_     BOOLEAN PersistData,
_In_opt_     PCWNF_TYPE_ID    TypeId,      // This is an optional way to
get type-safety
_In_     ULONG    MaximumStateSize,        // Cannot be above 4KB
_In_     PSECURITY_DESCRIPTOR     SecurityDescriptor  // *MUST* be present
);
```

כדי למחוק את ה-State Name ניתן להשתמש ב-ZwDeleteWnfStateName. כדי להוסיף Data ל-State Name ולשלוח התראה, אפשר לערוך State Name בעזרת ZwUpdateWnfStateData:

```
NTSTATUS ZwUpdateWnfStateData (
_In_     PCWNF_STATE_NAME         StateName,
_In_reads_bytes_opt_(Length)         const VOID* Buffer,
_In_opt_     ULONG    Length,      \\ Must be less than MaximumSize when
registered
_In_opt_     PCWNF_TYPE_ID    TypeId,      \\ Optionally, for type-safety
_In_opt_     const    PVOID       ExplicitScope,   \\ Process handle, User
SID, Session ID
_In_     WNF_CHANGE_STAMP     MatchingChangeStamp, \\ Expected current
change stamp
_In_     LOGICAL          CheckStamp   \\ Enforce the above or silently
ignore it
);
```

כדי למחוק Data ניתן להשתמש ב-ZwDeleteWnfStateData. כדי לקרוא State Data יש את ZwQueryWnfStateData:

```
NTSTATUS     ZwQueryWnfStateData(
_In_     PCWNF_STATE_NAME         StateName,
_In_opt_     PCWNF_TYPE_ID    TypeId,
_In_opt_     const VOID*      ExplicitScope,
_Out_        PWNF_CHANGE_STAMP    ChangeStamp,
_Out_writes_bytes_to_opt_(*BufferSize, *BufferSize) PVOID    Buffer,
_Inout_      PULONG       BufferSize   \\ Can be 0 to receive the current
size
);
```

בהרשמה ל-State Name קיים אתה יכול לקבל התראה כל פעם שמידע מפורסם לתוך ה-State Name, או כשה-State Name נהרס ואפילו כשמישהו אחר עושה לו Subscribe.

כדי להירשם ל-State Name יש את הפונקציה:

```
NTSTATUS RtlSubscribeWnfStateChangeNotification (
_Outptr_     PWNF_USER_SUBSCRIPTION*     Subscription,
_In_         WNF_STATE_NAME      StateName,
_In_         WNF_CHANGE_STAMP    ChangeStamp,
_In_         PWNF_USER_CALLBACK  Callback,
_In_opt_     PVOID           CallbackContext,
_In_opt_     PCWNF_TYPE_ID   TypeId,
_In_opt_     ULONG   SerializationGroup,
_In_opt_     ULONG   Unknown);
```

בנוסף לפונקציות שתיארתי בחלק הזה, שהן כולן User-Modeיות (למרות שהן Undocumented וגם מייקרוסופט לא מפרסמים את הסימבולים שלהם), ישנם גם כמובן פונקציות מקבילות אליהן ב-Kernel Mode:

```
ExSubscribeWnfStateChange
ExQueryWnfStateData
```

שיטות אפשריות לתקיפה

לאחר שסקרנו את המנגנון, והבנו קצת מה השימושים שלו ומה ניתן להשיג באמצעותו, נראה איך ניתן לנצל אותו לפעולות זדוניות. אך לפניי שנסקור את כל האפשרויות, יש 2 נקודות שחשוב לציין בנקודה זו בנוגע לשימוש ב-WNF בכללי.

ראשית, חשוב לזכור ש-WNF קיים רק החל מ-WIndows 8, כך שגם אם פוגען ייצרה להשתמש ב-WNF, הוא לא יעבוד על Windows 7.

נקודה שנייה היא ש-WNF, כמו כל מה שלא מתועד ב-Windows, יכול להשתנות בין כל עדכון של מערכת ההפעלה. כך שבעתיד אולי הפונקציה שבה השתמשנו מצפה לפרמטרים שונים, המיקום של המבנים בזיכרון השתנה או שבכלל השתנה מבנה ה-State Name, והקוד יפסיק לעבוד.

מעבר לכך שכל Administrator יכול לכתוב על גבי State Names בצורה כזו שתהרוס את מערכת ההפעלה, ישנם עוד בעיות אבטחה רבות במנגנון הזה.

קודם כל המימוש של המנגנון עצמו מלא בחורי אבטחה - במחקר של Alex Ionescu הוא סיפר על כך שהוא מצא בעיות אבטחה רבות כשהוא שיחק עם WNFDump, לדוגמא הרבה מידע שאמור להיות מוגבל בין תהליכים ובין משתמשים, ב-WNF הוא גלובאלי. את בעיות האבטחה האלו Microsoft יתקנו בגרסא הקרובה.

שיטות תקיפה בעזרת יצירת State Names חדש

שימוש ב-State Names יכולה להיות דרך טובה לשמירת מידע בדרך שהיא חשאית, ויכולה להחליף כל מיני שיטות שבהם פוגענים משתמשים. אחת התכונות החשובות של State Names היא כאמור האפשרות שלו להיות Persistent לאחר reboot. רישום של State Names חדשים מסוג Permanent או Persistent יכולה להיות דרך מעולה לשמור על Persistency במערכת.

אם ניצור State Names שהם Permanent או Persistent, ניתן יהיה לראות קריאה וכתיבה לרג'יסטרי אך זה יהיה הקרנל שיבצע אותם. כן יהיו אירועי ETW על הרשמה ל-State Name וביטול הרשמה, וגם על כתיבה ל-State Name. אך גם האירועים האלה נוצרים רק כשמשתמשים בפונקציות של Rtl. אם נשתמש ישירות ב-Zw, נוכל לעקוף את זה ולא יווצרו בכלל אירועי ETW.

מנגד, אם ניצור Temporary Name חדש, נאבד את ה-Persistency, אך לגמרי נמנע מהרג'יסטרי. כן צריך כאמור לקרוא ל-ZwCreateRegisterName, אך לא נשמר על זה תיעוד במערכת ההפעלה.

זאת בעצם דרך בלתי נראית להעברת מידע בין תהליכים. הדרך היחידה לגלות אותו היא אם יש לך Hook על הפונקציה הזאת.

היתרון הגדול של השימוש ב-WNF הוא בכך שכל הפעילות שלו (כמו הכתיבה לרג'יסטרי) מתבצעת ע"י עבודה פנימית של מערכת ההפעלה. בנוסף הוא יכול לתת לך Persistency. והכי חשוב - כלי ה-EDR היום לא מכירים את נתיבי הרג'יסטרי האלה, ובטח שלא יכולים להתחקות אחרי מי ביצע את הפעולה, והאם הפעולה היא לגיטימית או לא. כל אלה, ביחד עם עצם זה שהמנגנון נותן פלטפורמה מאוד נוחה להעברת מידע בין תהליכים ולשלוח מידע ל-Kernel והפוך, גורמים ל-WNF להיות פלטפורמה מאוד מושכת לכותבי פוגענים.

חלק מהשימושים בה יכולים להיות כדי להסתיר מידע שצריך שהוא יהיה Persistent. (אולי לשמור שם את ה-Stage2?) או כדי להעביר מידע בין תהליכים שונים בצורה מאוד חשאית בעזרת Temporary State Name.

שימוש ב-State Names "ריקים"

במערכת ההפעלה יש הרבה Well-Known State Names שלא בשימוש, לדוגמא כל אלה שקשורים ל-Xbox או ל-Windows Phone. (השמות שלהם מתחילים ב-WNF_XBOX ו-WNF_CELL בהתאמה)

כך שבמקום ליצור State Names חדשים כדי להסתיר בהם מידע, ניתן לעשות זאת גם דרך פרסום מידע ב-State Names הקיימים, שאנו יודעים שמערכת ההפעלה לא תיגע בהם. כאשר משתמשים ב-State Name- קיים כדאי למצוא אחד שנותן לכולם אפשרויות כתיבה וקריאה, ויכול להכיל עד 4KB.

מציאת חולשה של מערכת ההפעלה בפירסור ה-State Name

מכיוון שכל התראת WNF יכולה להכיל עד 4KB, ובגלל שזה מנגנון פנימי, התהליכים שמשתמשים במידע הזה בדרך כלל יפרסרו אותו בלי לבדוק את לגיטימיות הקלט, כך שייתכן שיש אפשרות שניתן לנצל חולשה ולגרום להזרקה של Shellcode משלנו.

שיטה אחת למצוא חולשה כזאת היא לעשות Fuzzing של הערכים בתוך ה-State Names ולראות מה גורם למערכת להתנהג באופן לא צפוי. (לדוגמא קריסה של Service או אפילו Blue Screen), שיטה אחרת היא דרך בחינה של ה-Subscriptions של תהליך, ולבדוק אותם ספציפית.

כדי למצוא את כל ה-Subscriptions של תהליך, יש להשיג את האובייקט nt!_Eprocess.WnfContext המכיל את האובייקט WNF_PROCESS_CONTEXT, שהוא אובייקט לא מתועד.

ניתן להשיג את האובייקט גם דרך nt!ExpWnfProcessesListHead, אשר מכיל רשימה מקושרת של ה-WNF_PROCESS_CONTEXT של כל התהליכים. האובייקט WNF_PROCESS_CONTEXT לא מתועד, אך אלכס הציג אותו בהרצאה שלו:

```
Typedef struct _WNF_PROCESS_CONTEXT
{
      WNF_CONTEXT_HEADER Header;
      PEPROCESS Process;
      LIST_ENTRY WnfProcessListEntry;
      Struct
      {
            PWNF_SCOPE_INSTANCE WnfScopeTypeSession;
            PWNF_SCOPE_INSTANCE WnfScopeTypeUser;
            PWNF_SCOPE_INSTANCE WnfScopeTypeProcess;
      } ImplicitScopeInstances;
      EX_PUSH_LOCK NameInstanceListLock;
      LIST_ENTRY TemporaryNameslistListHead;
      EX_PUSH_LOCK SubscriptionListLock;
      LIST_ENTRY ProcessSubscriptionListHead;
      EX_PUSH_LOCK PendingQueueLock;
      LIST_ENTRY DeliveryPendingListHead;
      PKEVENT NotificationEvent;
} WNF_PROCESS_CONTEXT, *PWNF_PROCESS_CONTEXT;
```

בתוכו יש כ-LIST_ENTRY את רשימת ה-Subscriptions של התהליך. אם רוצים לדעת מה דרייברים עושים עם המידע שהם מקבלים מה-State Names, ה-Subscriptions של התהליך System מכיל אותם. וכך אתה יכול להשיג את כל ה-Kernel Callbacks.

בדרך הזו אפשר לחפש דרייברים שלא משתמשים נכון במידע שהם מקבלים, ואולי למצוא דרך להרצת קוד בקרנל.

הזרקת קוד עם WNF

במקום השיטה הרגילה להזרקת קוד, שמשתמשת בפונקציות WriteProcessMemory, VirtualAllocEx, CreateRemoteThread, שכל כלי ה-EDR יודעים היום לזהות כפעילות עוינת, יש צורך למצוא שיטות שונות לעשות את אותו תהליך מבלי שזה יהיה מזוהה.

בשנה שעברה פורסמה שיטה בשם AtomBombing, שהראתה איך ניתן להזריק קוד לתהליך אחר בדרך יצירתית יותר - https://www.digitalwhisper.co.il/files/Zines/0x4E/DW78-3-AtomBombing.pdf.

דבר דומה יכול גם להיות אפשרי דרך WNF. אם ידוע על תהליך מסוים שקורא מתוך State Name מסוים, ע"י שינוי הערך ב-State Name הזה נוכל להכניס מידע משלנו לתוך זיכרון תהליך היעד.

לאחר שהמידע שלנו נמצא בזיכרון של תהליך היעד, צריך לגרום לשינוי הרצת התוכנית כדי שתריץ את הקוד שהזרקנו.

כדי לעשות את זה יש למצוא את פונקציית ה-Callback איתה הוא נרשם לאותו State Name, בדרך שהסברנו מקודם (דרך WNF_PROCESS_CONTEXT).

ברגע שמצאנו את זה, ניתן לשנות את ה-Callback Function, לפונקציה אחרת. למה שאנחנו הזרקנו.

CasperStager

לאחר שההרצאה פורסמה, כצפוי התחילו ליישם את השיטות האלו. כרגע יש רק PoC ראשון שנכתב ב-C#, בשם CasperStager, המממש חלק מהטכניקות שדיברנו עליהם:
https://github.com/ustayready/CasperStager

השיטה כאן היא שמירת ה-Stage2 של הפוגען בתוך State Name קיים, כמו שהצגתי בתחילת המאמר. (בגלל גודל התוכנית הוא שמר אותה ב-3 State Names שונים שקשורים ל-Xbox)

ה-Dropper בעצם מקבל מהאינטרנט את ה-Stage2 ורושם אותו אל תוך ה-State Name, במקום לשמור אותו על הדיסק. ואז בכל ריצה של התוכנית, קורא אותו משם ומריץ אותו.

זה לא נותן לך הרצה עצמאית של ה-Stage2, אך זאת דרך חמודה להסתיר את הפוגען "האמיתי", ואת הפעולות שלו.

הפעילות של ה-Stage2 בפרוייקט הזה היא למלא את כל ה-WNF State Names באפסים, כדי להשבית את המערכת.

סיכום

WNF הוא עדיין בגדר אדמה לא מיושבת, המידע והמקורות לנושא עדיין דלילים. למרות שעדיין אין מימוש של כל הטכניקות שהצגתי המנצלות את ה-WNF, כבר עכשיו ניתן לראות את ה"פוטנציאל" הטמון בו, ואני מאמין שבעתיד ניראה עוד ועוד פרוייקטים שמצליחים לנצל אותו לשימושים שונים.

פוגענים שישתמשו בשיטות האלה יהנו מכך שהמנגנון הזה גם כל כך גלובאלי ועדיין כל כך מתחת לרדאר של רוב פתרונות האבטחה היום.

למרות שנדמה לנו שאנו מכירים טוב את מערכת ההפעלה Windows, תמיד נוספים אליה פיצ'רים חדשים שיכולים להוות חור אבטחתי, והמירוץ חתול ועכבר בין כותבי הפוגענים למערכות ההגנה תמיד נמשך.

לכל שאלה או הערה על WNF או סתם בכללי ניתן ליצור קשר בכתובת blum.tal2@gmail.com

מקורות

[1] - ההרצאה מ-BH2018:

https://www.youtube.com/watch?v=MybmgE95weo

[2] הבלוג של RedPlait:

http://redplait.blogspot.com/2017/08/wnf-ids-from-perfntcdll.html

[3] רשימה של Well Known State Names:

https://github.com/ionescu007/wnfun/blob/master/script_python/WellKnownWnfNames.py

[4] דוגמא נהדרת שמסבירה על WNF:

https://blog.quarkslab.com/playing-with-the-windows-notification-facility-wnf.html

[5] https://processhacker.sourceforge.io/doc/ntzwapi_8h.html