SECURITY ARCHITECTURE IMPROVEMENTS IN WINDOWS 8.1

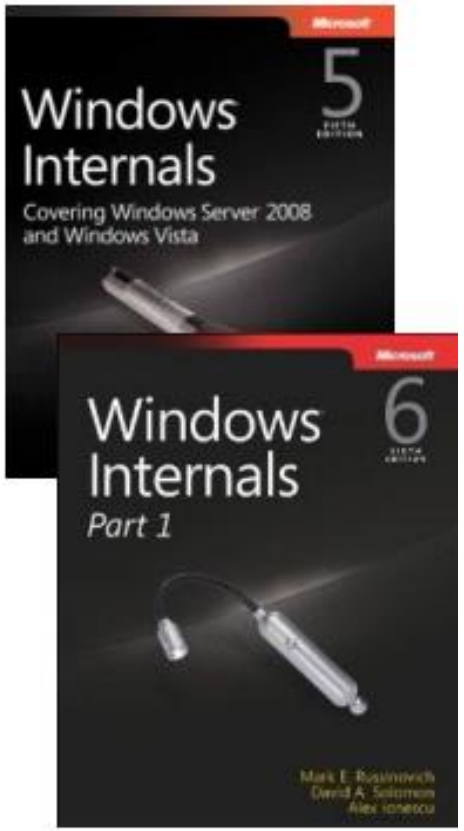
Alex Ionescu

ITCAMP 2014

Level 400

ABOUT ALEX IONESCU

- Chief Architect at CrowdStrike, a security startup
- Previously worked at Apple on iOS Core Platform Team
- Co-author of *Windows Internals 5th and 6th Editions*
- Reverse engineering NT since 2000 – was lead kernel developer of ReactOS
- Instructor of worldwide Windows internals classes
- Author of various tools, utilities and articles:
  - NTFS On-Disk Structure, Visual Basic Metadata Format
- Conference speaking:
  - SyScan 2012, 2013, 2014
  - NoSuchCon 2013, Breakpoint 2012
  - Recon 2014, 2013, 2012, 2011, 2010, 2006
  - Blackhat 2013, 2008
- For more info, see www.alex-ionescu.com



INTRODUCTION

- Windows 8.1 introduces core changes to the kernel that allow so-called "Anti-Malware" developers to better defend against local attackers
- Digital signatures and code signing now add an additional boundary of protection beyond load/don't load
  - Similar to the iOS Entitlement Model
- Mechanisms change a few core security paradigms:
  - Admin == Kernel is something that Microsoft has sometimes disagreed with, especially in light of PatchGuard, Code Signing and DRM
    - Now there is an official boundary
  - Unkillable processes and unstoppable services are now something supported and documented for developer (mis)use

OUTLINE

- Introduction
- Code Signing 101
- Signature Levels
- Process Protection
- Service Protection
- 3rd Party Support
- Runtime Signers
- Conclusion

CODE SIGNING 101

AUTHENTICODE

- For almost two decades, Portable Executable format (PE) supports Digital Code Signing through the Microsoft Authenticode standard
- Digital certificate (X.509 standard) contains a signature that ties the application to a publisher (or "signer") that is identified by a public key.
- Public key is used to compute a hash of the executable
- Hash is compared to the hash in the certificate itself
- Over the years, extended with
  - Embedded catalog support (to avoid bloating executable size)
  - Support for page hashes (so that each page can be hashed at runtime)
  - Stricter requirements (now SHA-256 is used and keys are 2048-bit or more)

ENHANCED KEY USAGE

- X.509 certificates are used by different applications for different *intended purposes*
  - SSL Encryption
  - EFS Encryption
  - Java Applet Signing
  - Windows Kernel Mode Driver Signing
  - …etc
- The "EKUs" in a certificate, which are requested during the certificate creation through the CA process, allow the signature validation algorithm to make sure that the certificate is being used in accordance to its issuance rules
  - For example, if a Root CA issued a 19$ SSL Certificate, it should not be used in order to authenticate a Windows kernel-mode driver load

WINDOWS EKU USAGE

- Windows traditionally only used one EKU to validate code signing requirements:
  - Code Signing (1.3.6.1.5.5.7.3.3)
- A few other EKUs were used for WHQL and determining OS files
  - Microsoft Publisher (1.3.6.1.4.1.311.76.8.1)
- Now this has changed with additional support for ELAM:
  - Early Launch Antimalware Driver (1.3.6.1.4.1.311.61.4.1)
- And new Windows 8.1 PPL support:
  - Windows System Component Verification (1.3.6.1.4.1.311.10.3.6)
  - Protected Process Light Verification (1.3.6.1.4.1.311.10.3.22)
  - Windows TCB Component (1.3.6.1.4.1.311.10.3.23)

CERTIFICATE DEMO

SIGNATURE LEVELS

SIGNING LEVEL NUMBER

- In Windows 8, the system needed to differentiate between Windows, Microsoft, and 3rd party signed files, instead of the black and white "signed or unsigned" choice
  - A system of integers was added in which each successive higher level indicates an increased amount of trust for the signer
- 0 -> Unsigned, 4->3rd party, 8->Microsoft, 12->Windows
- In Windows 8.1, this was extended to differentiate against the new types of processes and their protection levels
  - 6->Store, 7->Anti Malware, 14->Windows TCB
  - Other custom levels can be defined with a *signing policy*
- Signing level is granted based on the root key, the EKUs, and the operating system usage/requirement

SIGNING LEVEL CHECKS

- On Windows 8.1, signing levels are computed only for user-mode binaries that are run as protected processes
- On Windows 8.1 RT, they are computed anytime a process launches, based on SeILSigningPolicy
  - Defaults to 8, can be configured through different signing policy
- Process launch is not allowed if the User-Mode Code Integrity (UMCI) runtime does not allow the given certificate to grant the required signing level
  - This is called the Exe Signature Level
- Each successful process launch also carries with it the Dll (or Section) Signature Level
  - Image load is not allowed if UMCI runtime does not allow the certificate of any loaded DLL to grant the required section signing level

LEVELS DEMO

POLICY DEMO

PROCESS PROTECTION

VISTA PROTECTED PROCESS

- In Windows Vista, the term *protected process* was first introduced as a mechanism to protect the DRM requirements of the system
  - "Protected Media Path", PMP requires trusted components all the way to the output source
- Protected Processes had one bit set in the kernel EPROCESS structure at process launch time, if
  - CREATE_PROTECTED_PROCESS was passed to CreateProcess
  - Correct Windows Media DRM Certificate is present
- Media Foundation API took care of implementation details
  - Protected Environment Authorization (Peauth.sys) handles the obfuscation, key exchange, etc…
- *PsIsProtectedProcess* is documented for kernel-mode drivers and *NtQueryInformationProcess* can be used to check the PEB field as well

PROTECTED PROCESS RIGHTS

- Once a process runs with the bit set, all of its threads and processes are protected against accesses except:
  - PROCESS/THREAD_QUERY_LIMITED_INFORMATION
  - PROCESS_TERMINATE
  - PROCESS/THREAD_SUSPEND_RESUME
- Only other protected processes have their usual level of access based on ACL
- Debug privilege does not carry any weight here!
- In-box processes that run protected include
  - Mfpmp.exe (if playing protected media with WMP)
  - Audiodg.exe (Windows Audio Service Device Graph)

WINDOWS 8

- Minor change introduced in Windows 8, the ProtectedProcess bit disappears
- OR'ing bit 1 to the SignatureLevel makes the process protected
  - Thus, Audiodg.exe is 5 on Windows 8, 9 on Windows 8 RT
- Continues to be a DRM-enforcement mechanism only, however
- Note that System process also runs as Protected, however
  - This prevents Administrators from accessing the handle table or user-mode virtual address space of the process
  - ASLR relocation information is stored in user-mode
  - Some DRM/crypto temporary data as well
  - Handles could be duplicated and lead to access to protected kernel resources, such as hibernation file, page file, registry hives and transactional logs

WINDOWS 8.1

- Adds a new “Protection” field to EPROCESS composed of three elements
  - Protected Signer
  - Protected Type
  - Auditing Mode
- The Protected Signer is one of the root keys accepted by UMCI, possibly combined with one or more required EKUs
  - PsProtectedSignerAuthenticode = 0n1
  - PsProtectedSignerCodeGen = 0n2
  - PsProtectedSignerAntimalware = 0n3
  - PsProtectedSignerLsa = 0n4
  - PsProtectedSignerWindows = 0n5
  - PsProtectedSignerWinTcb = 0n6

PROTECTED PROCESS LIGHT

- The Protected Type is one of:
  - PsProtectedTypeProtectedLight = 0n1
  - PsProtectedTypeProtected = 0n2
- Combined together this creates a number such as 0x31
  - Anti-Malware Protected Light
- New API introduced: *PsIsProcessProtectedLight*
- Protected processes can only be accessed by Protected processes
- Protected Light processes can only be accessed by Protected Light or higher processes
  - Each signer dominates the other and creates an access hierarchy
- Different signers receive different access mask protections
  - See RtlProtectedAccess array (used by *RtlTestProtectedAccess*)

SYSTEM EFFECTS

- LSA processes, Anti-Malware processes, and Windows TCB processes can no longer be killed
  - But suspension is still possible
- Deprecates previous "Critical Process" functionality used to protect CSRSS and SMSS
- Makes Windows 8 RT public jailbreak impossible to execute, as it requires injection of code into CSRSS
- Makes injection/memory-based techniques for dumping LSA hashes, passwords, and secrets impossible
  - But only if LSA protection is enabled
- Also makes it that much harder to steal (impersonate) a SYSTEM token

PROTECTION ATTRIBUTE

- In Vista, CREATE_PROTECTED_PROCESS was the only flag needed for *CreateProcess* to do the right thing
- But in Windows 8.1, how to specify the actual protection level required (type and signer?)
  - Using the new Protection Level Attribute (0x2000B) in the Process/Thread Attribute List
- At the Win32 level, this is based on an undocumented enumeration:
  - 1 – Windows Signer, Protected Type
  - 2 – Windows Signer, Protected Light Type
  - 3 – Antimalware Signer, Protected Light Type
- At the NT level, it is converted into the actual PS_PROTECTED_SIGNER and PS_PROTECTED_TYPE number we saw earlier

PROTECTION CHAIN

- Recall that Windows 8.1 only checks signature levels if and only if a process is created requesting a protection level
  - Double-clicking a file on Explorer doesn't do this
- Therefore, there exists an implied "protection chain" that makes the system work:
  - The System process marks itself as protected
  - It launches SMSS requesting protection 0x61
  - SMSS launches CSRSS requesting protection 0x61
  - CSRSS launches Wininit and Winlogon with no protection (for now…)
  - Wininit launches LSASS with protection 0x41 (if configured)
  - Wininit launches SCM with protection 0x61
- But what about Anti-Malware?

LEVELS DEMO

PROTECTION DEMO

SERVICE PROTECTION

PROTECTED SERVICES

- Since Services.exe (aka the SCM/Service Control Manager) launches highly protected, this might imply that it can manage protected light processes…
- Since Anti-Malware applications run protected, this implies something must have had launched them requesting a protection level…
- Indeed, the key to Process Protection Light is that it must also allow 3rd party and/or non-System processes to also run protected
  - This is done by enhancing services to support a protection level of their own
- In Windows 8,1 SERVICE_CONFIG_LAUNCH_PROTECTED can be sent to *ChangeServiceConfig2*, with a matching SERVICE_LAUNCH_PROTECTED_INFO structure and level
- The SCM sends this protection level request in the *CreateProcess* call that it makes

PROTECTED SERVICES

- The SCM API ultimately ends up setting the key "LaunchProtected" in the Services database entry in the registry
- The following calls/commands immediately become blocked by non-protected callers:
  - *ChangeServiceConfig(2)* / sc config
  - *ControlService(Ex)* / sc start, stop, pause
  - *DeleteService* / sc delete
  - *SetServiceObjectSecurity* / sc privs
- Check is done by the RPC Server inside SCM, using *RtlTestProtectedAccess* and by asking the kernel for the caller's protection level
- AppXSvc, sppsvc, WdNisSvc, WinDefend and WSService run protected in Windows 8.1
  - Can check with "sc qprotection"

PROTECTED APPLICATIONS

- Therefore, services ultimately are in charge of the last chain of protected process launches as they can use *CreateProcess* to spawn children that are also protected
- Recall that the enumeration is undocumented, however!
- Microsoft expects developers to thus use PROTECTION_LEVEL_SAME, which is documented as the only valid type for PROC_THREAD_ATTRIBUTE_PROTECTION_LEVEL
- In this way, Anti-Malware services will spawn Anti-Malware processes, etc…
- User-launched applications will never run protected, unless the application employs a type of self-launcher and/or COM activation launcher method and requests a protection level for its spawned clone

PROTECTED SERVICES DEMO

REGISTRY PROTECTION DEMO

3RD PARTY SUPPORT

3RD PARTY SERVICES

- The first way for a 3rd party developer to access the PPL machinery is to use the documented *ChangeServiceConfig2* API and set a protection level
- However, CI will normally not grant protected status except if:
  - The EKU for "Process Protected Light Verification" is "Process Protected Verification" is present in the certificate
  - The binary is signed by one of the accepted root keys for the requested level (such as Microsoft's key)
- However, for this specific scenario, CI has code to also check against a list of *runtime signers*
  - Includes a list of other root keys to accept
  - As well as associated EKUs
- How to join this list?

ELAM

- Early Launch Anti Malware was introduced in Windows 8 in order to allow 3rd party AV Vendors earlier access to the boot cycle and boot driver loads
  - Can see hardware state/measured TPM information and can invalidate PCRs
  - Can see whenever a boot driver is being loaded and can veto
    - Some in-box drivers cannot be vetoed (“core drivers”)
- Access to signatures for white/blacklisting is done through registry
- ELAM drivers require a special signature from Microsoft
  - Requires 1+ year as a security company in good standing
  - Requires small footprint (< 128 KB) and latency (< 0.5 ms) during load callbacks
- Must unload after boot drivers have been started
- Implemented as Load Order Group: “Early-Launch”
- *IoRegister/UnregisterBootDriverCallback*

ELAM CERTIFICATE INFO

- In Windows 8.1, ELAM drivers can now have a MicrosoftElamCertificateInfo resource with a MSElamCertInfoID section
- This section can specify up to 3 different certificate hashes, each with up to 3 EKUs, that can be used to augment CI's runtime signature database
- When an ELAM driver loads, Windows now scans the resource section for this table and registers with CI the contents within
- When a protected process launch is performed, and a required signing level is sent to CI, the runtime-registered signers database is checked to see if the certificate matches one of the hashes, and if the EKUs match any of the EKUs registered for the hash

ELAM EKU DEMO

ELAM CERTIFICATE INFO DEMO

RUNTIME SIGNERS

NON-BOOT ELAM SIGNATURES

- Because Windows 8.1 only loads ELAM drivers at boot, CI will only receive the list of runtime signers if the ELAM driver is already present
- What if the vendor wants to provide a user-mode PPL service from the start, without having to wait for a reboot?
  - What if the vendor does not actually want to implement any boot-time ELAM functionality?
- A new API, *InstallElamCertificateInfo* can be used along with a file handle to an ELAM driver, which will do the same boot-time parsing
  - In fact, it doesn't really have to be an "ELAM Driver", as long as the file is signed with the appropriate ELAM EKU
  - Internally calls an undocumented *NtSetSystemInformation* class
- Ultimately calls *CiRegisterSigningInformation*
  - There is a matching Unregister call, but no documented way to call it

NON-ELAM SIGNATURES

- *CiRegisterSigningInformation* actually doesn't enforce any limit on the number of certificate hashes and/or EKUs that can be registered
  - Done by the kernel while handling the user-mode API call
- Additionally, the ELAM validation check is also done by the kernel during the API call
- In other words, kernel callers can register their own runtime signers outside of ELAM regulations
- So although the model suggests that an Admin must obtain an ELAM-signed Driver from Microsoft, which can then register custom 3rd party PPLs…
- … the reality is that an Admin can obtain **any** signed Driver from Verisign/Digicert/etc, which can then register custom 3rd party PPLs
  - And on 32-bit systems, the driver doesn't even have to be signed!

RUNTIME SIGNERS LIST DEMO

CONCLUSION

- Protected Processes in Windows 8.1 create a few headaches for IT Administrators and Power Users
  - Can no longer kill certain processes
  - Can no longer debug certain processes
- Security software developers (and anyone that can obtain an ELAM certificate) can leverage the functionality to protect their application
- By calling *SeRegisterSigningInformation* any software developer, even non-security, can leverage this functionality by having a driver call the function
  - However, requires relying on undocumented functionality which may be removed
- On the flip side, security and other memory-scanning/forensic-type application developers now have to deal with impenetrable processes unless they write a driver that bypasses the protection

BONUS 1: DISABLING PROTECTION

BONUS 2: ENABLING LSA AS A PPL

BONUS 3: SE IMAGE CALLBACKS

- *PsSetLoadImageNotifyRoutine* is the API through which antivirus drivers are able to get notifications about driver loads
  - Blocking the driver load at this point is not obvious – function has no support for this
  - Must rely on other tricks, such as overwriting entrypoint with a stub that returns failure, or attempt to suspend the load
  - Or rely on file system filter and block access at the source
- ELAM provides new API for antivirus drivers which allows blocking, but only works for boot drivers
- Windows 8.1 introduces new *SeRegisterImageVerificationCallback* API
  - Provides same data as ELAM (driver signature, hash, fingerprint)
  - Provides ability to allow or block the driver load