Defeating Pass-the-Hash

Separation of Powers
Credential Theft

• At the heart of many high-profile attacks.

• Fueled by Single Sign-On
  • A feature nobody wants to live without.
Pass-the-Hash: A Windows Primer

- LSASS on Alice’s laptop hosts the authentication protocols

- Administrator-level attackers may access:
  - NTLM Hash
  - Kerberos Keys
  - Alice’s password

- Attackers steal and replay these legacy protocol artifacts
The Chain Reaction

Foothold compromise: attacker harvests creds

Using the creds, attacker spreads to other devices, mining more creds.

Repeat
We Have the Technology

• Multi-Factor Authentication
  • Stealing one credential isn’t enough.

• Strong Credentials
  • Smart cards, FIDO key, etc

• Token Binding
  • Make stolen tokens useless.
Businesses Like Making Money

• Legacy components keep working
  • “My printer works with NTLM.”
  • NAS, Printers, Software, etc.
  • Business depends on these

• Legacy protocols include replayable artifacts
How to keep a secret?
Separation of Powers

• Balance of powers prevents abuse
• Ensures accountability

• Legislation passes the laws.
• Executive branch carries out the tasks.
• Judicial system make sure everyone is playing by the rules.

• OS and real governments aren’t that different.
• Administrators → The Legislative Power
• Kernel / System Services / Drivers → Executive Power
• Trusted Computing Base (TCB) → Judicial Power (makes sure everyone obeys the constitution)
Admin == Kernel == TCB: Risky business

- **Admins are human, humans err**
  - Data shows: > 90% (!!) of Windows users run as some sort of administrator
  - Total loss of system when a malicious attachment is run

- What if the **administrator is malicious**?
  - Admins should not have total control on the machine
  - E.g. games, multi-tenant scenarios

- We can’t simply trust the kernel, either.
  - Attack surface too big: Thousands of system calls, IOCTLs
  - Diverse ecosystem: Many 3rd party drivers with different quality assurance standards
This is not a new problem...

- **Authencode / Kernel Mode Code Signing**
  - Principle: Putting reputation of an authenticated identity on the line
  - Cost + traceability negatively impacts exploit economics
  - Problem: Strong verification of publishers by CAs is questionable at best and recalls are hard and slow.

- **Protected Process – PP / Protected Process Light – PPL**
  - Principle: Isolate sensitive processes from others by preventing injection of threads, memory access, etc.
  - Problem: Not enough, still vulnerable to kernel mode, which is not TCB.

- **Patch-guard**
  - Principle: Limit what code in kernel mode can do
  - Problem: Heuristic based, not failsafe

- **They are all software based...**
- **Can the security be rooted on something.. harder?**
Layers of protection via Hardware

- X86/X64 systems have had a single physical address space in kernel
  - Ring 0 could access any physical memory address.
  - Ring 0 → God Mode

- “Hypervisor” provided another abstraction layer
  - AKA Ring (-1)
  - Roots its promises on HW
    - Just like rings...
    - But hypervisor is small.. very small. Easier to verify, easier to secure.
  - Hypervisor is the true TCB

- We need hypervisor kind of isolation without cluttering hypervisor.
Introducing Virtual Trust Levels - VTL

• Using virtualization technologies and Second Level Address Translation (SLAT), sections of memory can be access-protected in a cascading fashion

• Guest virtual → Guest physical → System physical
### Introducing Virtual Trust Levels - VTL

<table>
<thead>
<tr>
<th>Ring</th>
<th>Address Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>User Address Space</td>
</tr>
<tr>
<td>0</td>
<td>Kernel Address Space</td>
</tr>
<tr>
<td>-1</td>
<td>Hypervisor</td>
</tr>
</tbody>
</table>

Less accessible

[Image of the diagram]
Introducing Virtual Trust Levels - VTL

VTLs bring a new dimension with new properties.
### Virtual Trust Levels - VTL

<table>
<thead>
<tr>
<th>Ring</th>
<th>VTL 0</th>
<th>VTL1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>User Address Space</td>
<td>User Address Space (inaccessible to VTL0)</td>
</tr>
<tr>
<td>0</td>
<td>Kernel Address Space</td>
<td>Kernel Address Space (inaccessible to VTL0)</td>
</tr>
<tr>
<td>-1</td>
<td>Hypervisor</td>
<td></td>
</tr>
</tbody>
</table>

**Less accessible**

- Regular Windows, “Normal world”, runs in VTL0
- “Secure world”, new in Windows 10 is **selectively inaccessible** to normal world, even normal NTOS.
  - Code can be safely shared / reused
  - Data can be shared so that VTL0 / 1 can pass data back and forth as needed
Introducing Virtual Trust Levels - VTL

- Unlike rings, VTLs are extensible

<table>
<thead>
<tr>
<th>Ring</th>
<th>User Address Space</th>
<th>Kernel Address Space</th>
<th>Hypervisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>User Address Space (inaccessible to VTL0)</td>
<td>Kernel Address Space (inaccessible to VTL0)</td>
<td>TBD</td>
</tr>
<tr>
<td>0</td>
<td>Kernel Address Space</td>
<td></td>
<td>TBD</td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Normal World – Pretty much as always

Normal World (VTL 0)

Normal User
- System services
- Apps

Threads

Normal Kernel
- NTOS
- Drivers

System Calls
- NT Services

Drivers

Memory Manager

Hypervisor & UEFI/TPM & HW
Introducing Secure World

Normal User
- System services
- Apps

Secure User
- (a.k.a. Isolated User Mode)

Normal Kernel
- NTOS
- Drivers

Secure Kernel
- Proxy kernel
- System Calls / Services

Hypervisor & UEFI/TPM & HW
Introducing Secure World

Normal User
- System services
- Apps
- Threads

Secure User (a.k.a. Isolated User Mode)
- Threads
- RPC

Normal Kernel
- NTOS
- Drivers
- System Calls
- NT Services
- Memory Manager

Secure Kernel
- Proxy kernel
- Secure Kernel System Calls / Services
- Memory Manager
- Marshaller / Unmarshaller Hardener

Hypervisor & UEFI/TPM & HW
Secure World

- **Invisible**
  - No user interaction / UI
  - Minimal impact on perf (< 5%)
- **Tighter control**
  - No 3rd party code in the secure kernel
  - Trustlets are isolated from each other
  - Trustlets are limited in number, purpose built - much smaller, easier to protect
- **World is small.. Secure world is smaller.**
  - If no secure mode, a trustlet can run as a normal mode process
  - Secure world relies on enlightened normal world / NTOS for many things (scheduling, most of memory management, synchronization etc.)
  - Secure kernel only does the bare minimum (configuring SLAT as applicable, encrypting pages before paging out, etc.)
  - VTL0 is not trusted → Secure kernel hardens its NTOS interfaces
Using VSM to Mitigate PtH

You can’t pass the hash if you don’t have it
Credential Strength

• Weak credentials are easily stolen by
  • Cookie Theft
  • Phishing
  • Key Logging

• Strong credentials are theft resistant
  • Smart card
  • Two factor authentication

• Users with weak credentials are vulnerable.
1. Prove identity and receive a Ticket Granting Ticket
2. Present TGT to gain a service ticket
3. Present service ticket to access service.

But wait! There’s more...
4. The service ticket reply contains an NTOWF for NTLM compatibility
Isolation Architecture

- LSASS continues to run in normal world
  - Core protocol logic stays in LSASS

- Cred Guard provides *isolation services* to LSASS
  - All use of secrets happens here

- LSASS talks to Cred Guard over RPC

- Secure-mode keys encrypt data
  - No clear secrets in normal world
Artifact Isolation

- Old: Everything in LSASS
  - Bad admin owns you
- New: All “passable” secrets protected by Cred Guard
  - Secrets are now hidden
  - Attackers cannot steal secrets from memory they cannot read.
- However... Attackers still have oracle access to the user’s credential.
  - We’re not there yet.
Ensuring Secrets are Isolated

• An attacker with oracle access to your cred can PtH

• Isolation is only good if we can guarantee it.
  • Client trickery is never enough.

• Solution: Kerberos FAST (RFC 6113)
  • Compound authentication: What machine is a user coming from.
  • Provides the promise of truly hidden artifacts
Foundation: Strong Machine Credentials

- Like users, systems have credentials.
  - Traditionally passwords
  - Key pairs are supported as of 2012 R2

- Cred Guard owns the system private key.
  - Attackers cannot access this credential.

- We combine this with compounding (FAST)
  - 2012 R2 allows binding of users to machines
    - Authentication policies
Compound Authentication

- Machine authentication uses an Cred Guard-protected ID Key.
  - The machine uses this to get a TGT

- A derived, armor key is created.
  - Alice combines her credential with the proof.
  - The KDC checks the proof and grants a TGT.

- Attackers have zero access to the machine ID key, preventing illicit authentication attempts.
The Path to Secure Users

- Secured users *only* use strong authenticators
  - Attackers cannot steal this authenticator.

- Secured systems authenticate with an ID key
  - Attackers have zero access to the machine ID key

- Secured users may authenticate only from designated systems
  - This policy is validated at the KDC.
What if I Turn it Off?

- What happens if the bad guy turns off Cred Guard?
- Alice, and the attacker, can still use the smartcard
- Without the proof of origin, the KDC denies the request for a TGT.
Demo Time
Steps to Mitigating PtH

• Eliminate weak protocols – MSCHAPv2, NTLMv1
• Migrate users to strong credentials
• Update hardware refresh specs to IUM-compatible devices
• Enable Win10 IUM support
• Get educated on other Credential Theft mitigations
  • http://www.microsoft.com/pth
VSM platform requirements

• Virtualization Extensions (Intel VT-x)
• Second Level Address Translation, SLAT (Intel Extended Page Tables, EPT)
• IOMMU (Intel VT-d)
• UEFI 2.3.1
• TPM 2.0
• Optional Performance Enhancement - Mode Based Execution Control (MBEC)
  • Optimal performance for CI enforcement
  • Fall-back to S/W based implementation