Pass-The-Hash Toolkit for Windows
Implementation & use

Hernan Ochoa
(hochoa@coresecurity.com, hernan@gmail.com)
I’m going to talk about..

- What is the ‘Pass-the-hash’ technique?
  » Brief history and explanation of the technique
    - Current (previous) Implementations and limitations

- What is the Pass-the-hash Toolkit for Windows?
  » Brief history
  » Description of included tools and advantages
  » Implementation (technical details)
  » A ‘new’ post-exploitation ‘attack/technique/thing to do’
  » How to use the tools
    - Demos
  » If someone is still in the room… Q/A.
What is the ‘Pass-the-hash’ technique?
What is Pass-the-hash?

- Windows stores, generally, two **hashes** of a user’s passwords in its ‘users database’ (e.g.: SAM)
  - LM hash, NTLM hash

- “Pass-the-hash” allows an attacker to use LM & NTLM hashes to authenticate to a remote host (using NTLM auth) without having to decrypt those hashes to obtain the cleartext password


- How/Why does ‘Pass-the-hash’ work?

(over)simplified diagram of NTLM challenge-response authentication protocol

Client connects to Server and sends features supported, requested, etc.

Server responds w/features supported, required and sends random challenge (C)

Client calculates and sends $R = F(LM/NTLM \text{ pwd hashes}, C)$ along with username, domain, etc.

Server calculates $R$ locally, compares with client $R$ and returns result

- How hashes are used (F) varies (ntlmv1,ntlmv2,etc)
- Having LM/NTLM Hashes == having the cleartext password for remote NTLM auth
How do you obtain the LM&NTLM hashes to ‘Pass-the-hash’?

- Post-Exploitation

- Dump SAM database using pwdump3/3e/4/5/6/7, fgdump, etc.

  » Administrator:
  500:0102030405060708090A0B0C0D0E0F10:0102030405060708090A0B0C0D0E0F10:::
How do you obtain the LM&NTLM hashes to ‘Pass-the-hash’?, cont. (2)

- From c:\windows\repair\sam
- From c:\windows\system32\config\SAM
- Sniff SMB challenge-response over the network
  
  - Simplifying: capture the nonce and encrypted nonce
    
    » Need to brute-force to obtain a hash to ‘pass-the-hash’ (e.g.: use l0phtcrack, cain&abel)
    
    » Common misconception is to believe the ‘encrypted nonce’ is a hash we can work with, but it is not.
  
- Cachedump to obtain ‘hashed’ hashes 😊 and then brute-forcing..
- Etc...
Available ‘Pass-the-hash’ implementations

- Paul Ashton’s original ‘exploit code’: modified SAMBA client
  
  » With cleartext-password (not actual `smbclient` params):
    
    - `smbclient //192.168.1.20/diskC –U Administrator –p mypwd`
      
      » Analog to ‘net use z: \192.168.1.20\diskC /u:Administrator mypwd’

  » The patch allows the following (not actual `smbclient` params):
    
    - `smbclient //192.168.1.20/disckC –U Administrator –p`
      4ECC0E7568976B7EAAD3B435B51404EE: 551E3B3215FFD87F5E037B3E3523D5F6
Available ‘Pass-the-hash’ implementations, cont. (2)

- Lots of impl. with the same approach since then:
  - Samba-TNG provides built-in functionality for ‘passing-the-hash’
  - Lots of third-party implementations of the NTLM authentication mechanism allow performing the ‘pass-the-hash’ technique
    - In python, ruby, java, you name it..
    - Including metasploit, CORE IMPACT, impacket, etc.
Pass-the-hash previous implementations “limitations”

- Mostly, limited functionality:
  - Samba & Samba-TNG: enormous amount of functionality but still not everything is implemented
  - Other third-party libraries/programs implement even LESS functionality than Samba & Samba-TNG
  - Functionality is scattered among different libraries/programs
  - Some protocols and functionality is ‘partially implemented’

» Third-party implementations are always running behind:
  » Implementation is done by reverse-engineering and it takes a considerable amount of effort/time

» You can’t use native Windows tools
**What is Pass-the-hash Toolkit for Windows?**

- A set of tools that brings pass-the-hash to the Windows platform (and more)

- Published in 2007, is Free and Open Source (written in C, by me 😊)

- Currently, it works on Windows XP, Windows Server 2003 and Vista
What is Pass-the-hash Toolkit for Windows?, cont. (2)

- I first developed a fully-working version of this technique for Windows NT4 (and later for Win2000) in 2000:
  - I couldn’t publish the code back then (it was sold to a ‘company’)
  - But I wrote a paper: “Modifying Windows NT Logon Credentials”
    - Check out http://www.coresecurity.com/content/modifying-windows-nt-logon-credential

- In 2007, I wrote a completely new implementation of the technique from scratch and the PSH/PTH Toolkit was born
Pass-the-hash Toolkit for Windows memorabilia

- Pass-the-hash Toolkit for Windows memorabilia
PSH/PTH Toolkit for Windows Advantages

- Mainly, available functionality is “unlimited”
  - It run on Windows! So…
  - You can use any tool that uses NTLM authentication
    - from Microsoft or any other third-party tool (think admin interfaces, DCOM, etc)
  - You can use the same tools you’d use if you had the cleartext password
  - You have access to all available functionality and not partial implementations
  - You can use it on compromised remote Windows boxes during pentests and then use windows native tools
PSH/PTH Toolkit for Windows Advantages, cont. (2)

- PSH/PTH also provides a post-exploitation ‘technique/attack/tool’
  - ‘Steals’ credentials stored in memory
  - Using this, you may be able to own a windows domain more easily, more on this later..
Implementing ‘Pass-the-hash’ on Windows
What do we want to achieve?

- Analog functionality to `\smbclient //<server>/<share> -U Administrator -p 4ECC0E7568976B7EAAD3B435B51404EE: 551E3B3215FFD87F5E037B3E3523D5F6`:
  - Net use z: \\<server>/<share> -U Administrator 4ECC0E7568976B7EAAD3B435B51404EE:551E3B3215FFD87F5E037B3E3523D5F6
  - But for ALL tools that use Windows native support (API) for NTLM auth

- We want to be able to do it as many times as we want without logging in and out

- We want to do it without having to reboot the ‘attacking machine’
So, how do we do all that?

- Let’s take a look at the Windows NT Logon and Authentication model…
Three basic components take part

- **Logon processes**: a component trusted by the OS to monitor I/O devices for logon attempts

- **The LSA (Local Security Authority) Server Process**: user-mode process (lsass.exe) responsible basically for the local system security policy and user auth.

- **Authentication packages**: component (DLL) responsible for performing actual user’s credentials auth
  - Each auth pkg registers to the LSA at startup (authpkg id)
  - **Create new LSA Logon Sessions**
  - Return info for inclusion in Token object
    - The token represents security context for access
    - The auth packages associate credentials with the user’s logon session
Pass-The-Hash Toolkit For Windows
Winlogon.exe and msv1_0.dll

WINLOGON.EXE
(logon process)

NTLM AUTH

• Winlogon.exe: default logon process for interactive logons

• MSV1_0.DLL: NTLM auth package

• LSASS.EXE: keeps track of logon sessions

LSA AUTH API

MSV1_0.DLL
(auth pkg)

• ... (other auth pkgs)

LSASS.EXE
Winlogon

- Intercepts logon attempts from the keyboard

  - calls `LsaLogonUser()` with msv1_0’s id
    » This ends up in MSV1_0.DLL
- **Msv1_0**
  - Authenticates user using local sam or AD etc
    - Creates logon session (LUID)
  - **Msv1_0 adds credentials to logon session** by calling `LsaAddCredential()`
    - The username, the domain name, and the LM&NTLM hashes
  - These are the credentials used by windows when you try to access remote resources (e.g.: `net use \server \c$`)

**WINLOGON.EXE**
(logon process)

**LSA AUTH API**

**MSV1_0.DLL**
(auth pkg)

... (other auth pkgs)

**LSASS.EXE**
Msv1_0 communicates with LSA using the LSA AUTH API:

» Auth packages export the function

```c
NTSTATUS LsaApInitializePackage(
    __in      ULONG AuthenticationPackageId,
    __in      PLSA_DISPATCH_TABLE LsaDispatchTable,
    __in_opt  PLSA_STRING Database,
    __in_opt  PLSA_STRING Confidentiality,
    __out     PLSA_STRING *AuthenticationPackageName
);
```

» LSA calls this function at startup and passes the LsaDispatchTable structure
- **LSA_DISPATCH_TABLE**
  - Structure that contains the addresses of LSA functions that can be called by auth packages.

    typedef struct LSA_DISPATCH_TABLE {
        PLSA_CREATE_LOGON_SESSION CreateLogonSession;
        PLSA_DELETE_LOGON_SESSION DeleteLogonSession;
        PLSA_ADD_CREDENTIAL AddCredential;
        PLSA_GET_CREDENTIALS GetCredentials;
        PLSA_DELETE_CREDENTIAL DeleteCredential;
        PLSA_ALLOCATE_LSA_HEAP AllocateLsaHeap;
        PLSA_FREE_LSA_HEAP FreeLsaHeap;
        PLSA_ALLOCATE_CLIENT_BUFFER AllocateClientBuffer;
        PLSA_FREE_CLIENT_BUFFER FreeClientBuffer;
        PLSA_COPY_TO_CLIENT_BUFFER CopyToClientBuffer;
        PLSA_COPY_FROM_CLIENT_BUFFER CopyFromClientBuffer;
    } LSA_DISPATCH_TABLE, PLSA_DISPATCH_TABLE;
So, how can we implement ‘Pass-the-hash’ on Windows ALREADY!?

» We play around with the logon sessions and their associated credentials…
  – Remember…
  » Credentials associated with logon sessions are the credentials used when you want to access a remote resource using NTLM auth

» So if we change these credentials (e.g.: modify the password hashes), we modify credentials used for over the network auth and we will accomplish our goal
LSASS.EXE maintains a double-linked list of logon sessions

LSASS.EXE

SESSION_ENTRY

<table>
<thead>
<tr>
<th>Next</th>
<th>Prev</th>
<th>...</th>
<th>userLen</th>
<th>userPtr</th>
<th>DomainLen</th>
<th>DomainPtr</th>
<th>...</th>
<th>ptrToCreds</th>
</tr>
</thead>
</table>

Unk1 | AuthPkgId | PtrToCreds |

CREDS_ENTRY

<table>
<thead>
<tr>
<th>Unk1</th>
<th>PrimaryLen</th>
<th>PrimaryPtr</th>
<th>HashesLen</th>
<th>HashesPtr</th>
</tr>
</thead>
</table>

NTLM_CREDS_BLOCK (encrypted)

<table>
<thead>
<tr>
<th>DomainLen</th>
<th>DomainOffset</th>
<th>userLen</th>
<th>userOffset</th>
<th>NTLMhash</th>
<th>LMHash</th>
<th>...</th>
<th>DomainName</th>
<th>userName</th>
</tr>
</thead>
</table>

LSASRV.DLL!LogonSessionList (LSASRV.DLL!LogonSessionListCount)

LSASS.EXE maintains a double-linked list of logon sessions
Each logon session may have associated NTLM credentials (or others)
  - NTLM creds. encrypted w/random key using either desX-cbc or rc4
    » If \( \text{modulo(size/8)} \equiv 0 \) use desX-cbc, otherwise use rc4

  » DES-X (or DESX) is a variant of DES intended to increase the complexity of a brute-force attack using a technique called key whitening.
    - DES-X augments DES by XORing an extra 64 bits of key (K1) to the plaintext \textit{before} applying DES, and then XORing another 64 bits of key (K2) \textit{after} the encryption
- I’ve never seen credentials encrypted with rc4
- desX key appears to be lost but IV, whitening keys and scheduled key are available

  » LSASS itself uses this info to encrypt/decrypt
    - it uses the \texttt{LSASRV.DLL!LsaEncryptMemory()} function
LSASRV.DLL!LsaInitializeProtectedMemory generates the keys used to encrypt credentials in memory

```c
// global_vars
uchar *g_pRandomKey;
ulong g_cbRandomKey;
ulong CredLockedMemorySize;
void* CredLockedMemory;
_desxtable *g_pDESXKey;

unsigned __int64 g_Feedback;

LsaInitializeProtectedMemory
{
    g_cbRandomKey = 0x100 (256);
    CredLockedMemorySize = 0x190 (400);

    CredLockedMemory = VirtualAlloc(0, 190h, MEM_COMMIT(1000h), PAGE_READWRITE(4))
    VirtualLock( CredLockedMemory, CredLockedMemorySize );

    // _desxtable *g_pDESXKey
    g_pDESXKey = CredLockedMemory;
    g_pRandomKey = g_pDESXKey + 0x90 (144);

    SystemFunction036@8( g_pRandomKey, 0x18 (24) );
    SystemFunction036@8( &g_Feedback, 8);
    desxkey( g_pDESXKey, g_pRandomKey);
    SystemFunction036@8( g_pRandomKey, g_cbRandomKey );
}
```
LSASRV.DLL!LsaEncryptMemory is used to encrypt/decrypt credentials

```c
void LsaEncryptMemory(unsigned _int8 *buffer, unsigned __int32 len, unsigned int mode)
{
    char *pbuffer;
    ??? outRC4key;
    unsigned int feedback1;
    unsigned int feedback2;
    if( buffer == NULL) return;
    pbuffer = buffer;
    if( len == 0 ) return;
    if( !(len&7) ) {
        rc4_key( &outRC4key, g_cbRandomKey, g_pRandomKey);
        rc4( outRC4Key, len, buffer);
        return;
    }
    feedback1, feedback2 = g_Feedback;
    _CBC@28( &function_desX@16,
             8,
             buffer,
             buffer,
             g_pDESXKey,
             mode,
             &feedback1);
}
```
Pass-the-hash Toolkit for Windows
included tools & implementation
- **PSH/PTH Toolkit for Windows – included tools**

  - *IAM.exe* and *IAM-ALT.exe*: performs ‘pass-the-hash’

  - *WHOSTHERE.exe* and *WHOSTHERE-ALT.exe*: obtain credentials stored in memory (domain, username, NT&NTLM hashes)

  - *PASSTHEHASH.IDC*: *IDA Pro* .IDC script; obtain addresses *IAM.exe* and *WHOSTHERE.exe* need to function

  - *GENHASH.exe*: helper tool. Mainly for testing purposes:
    - Generates NT&NTLM hashes from a cleartext password
**GENHASH.EXE**

- Generates LM & NTLM hashes
- Uses ‘undocumented’ functions
  - `Advapi32.dll!SystemFunction006(strupr(char* pwd), out uchar* hash)`
    - Generates LM hash
  - `Advapi32.dll!SystemFunction007(unicode* pwd, out uchar* hash)`
    - Generates NTLM hash
The “hard” way

(iam.exe / whosthere.exe)
IAM.EXE and IAMDLL.DLL

- Findfuncs() in LSASRV.DLL
  - LsaAddCredential, LsaEncryptMemory, Feedback, DesXKey, LogonSessionList, LogonSessionCount

- Gets current LogonID
  - If –r, creates new logon session and process (advapi32.dll!CreateProcessWithLogonW)

- Creates ‘NTLM_CREDs_BLOCK’

<table>
<thead>
<tr>
<th>DomainLen</th>
<th>DomainOff</th>
<th>userLen</th>
<th>userOffset</th>
<th>NTLMhash</th>
<th>LMHash</th>
<th>...</th>
<th>Domain</th>
<th>User</th>
</tr>
</thead>
</table>

NTLM_CREDs_BLOCK

- Injects iamdll.dll into LSASS.EXE
  - Encrypts credentials manually and
    calls LSASRV.DLL!LsaAddCredential(LogonID,&primaryKey,&MSV_CREDs)
WHOSTHERE.EXE
- `Findfuncs()` inside LSASRV.DLL
  » LsaAddCredential, LsaEncryptMemory, Feedback, DesXKey, LogonSessionList, LogonSessionCount
- From LSASS.EXE
  » Reads value of g_Feedback, DesXKey, LogonSessionList, LogonSessionListCount
- Iterates thru items in double-linked list of sessions

**SESSION_ENTRY**

| Next | Prev | ... | userLen | userPtr | DomainLen | DomainPtr | ... | ptrToCreds |
- Gets to encrypted credentials per each logon session

```
<table>
<thead>
<tr>
<th>Unk1</th>
<th>AuthPkgId</th>
<th>PtrToCreds</th>
</tr>
</thead>
</table>

CREDS_ENTRY

<table>
<thead>
<tr>
<th>Unk1</th>
<th>PrimaryLen</th>
<th>PrimaryPtr</th>
<th>HashesLen</th>
<th>HashesPtr</th>
</tr>
</thead>
</table>

CREDS_HASH_ENTRY

<table>
<thead>
<tr>
<th>DomainLen</th>
<th>DomainOffset</th>
<th>userLen</th>
<th>userOffset</th>
<th>NTLMhash</th>
<th>LMHash</th>
<th>…</th>
<th>DomainName</th>
</tr>
</thead>
</table>

NTLM_CREDS_BLOCK (encrypted)
Findfuncs()

- **Address group**
  - LSASRV.DLL!LsaEncryptMemory@@YGXPAEKH@Z
  - LSASRV.DLL!_LsapAddCredential@16
  - LSASRV.DLL!g_Feedback@@3_KA
  - LSASRV.DLL!g_pDESXKey@@3PAU_desxtable@@A
  - LSASRV.DLL!LogonSessionCount@@3KA / LSASRV.DLL!LogonSessionListCount@@3KA (in W2003)
  - LSASRV.DLL!LogonSessionList@@3U_LIST_ENTRY@@A / LSASRV.DLL!LogonSessionList@@3PAU_LIST_ENTRY@@A (in W2003)
### Address Group Example

```c
#define V2976_XPSP2_ADDCREDENTIAL_FRENCH (PBYTE)0x756C7A24
#define V2976_XPSP2_ENCRYPTMEMORY_FRENCH (PBYTE)0x756C5449
#define V2976_XPSP2_FEEDBACK_ADDR_FRENCH (PBYTE)0x75750BD8
#define V2976_XPSP2_DESKEY_PTR_ADDR_FRENCH (PBYTE)0x75750BE0
#define V2976_XPSP2_LOGON_SESSION_LIST_ADDR_FRENCH (PBYTE)0x7574FCB8
#define V2976_XPSP2_LOGON_SESSION_LIST_COUNT_FRENCH (PBYTE)0x7574FE54
```
'Database' of 'addresses groups' for different LSASRV.DLL versions

- addresses change based on
  » DLL version of auth components
  » Service pack
  » Windows version (XP, 2003, etc)
  » Language (French, German, etc)
The “easy” way
(iam-alt.exe / whosthere-alt.exe )
IAM-ALT.EXE and PTH.DLL
- Gets LogonID
  » If -r, create new logon session and process (advapi32.dll!CreateProcessWithLogonW)
    - Obtain LogonID
- Injects PTH.DLL into LSASS.EXE
  » Finds msv1_0.dll!NlpAddPrimaryCredential
    - Not exported
    - Searches for signatures (series of fixed opcodes)
  » Calls msv_10.dll!NlpAddPrimaryCredential
    - No need to encrypt credentials
WHOSTHERE-ALT.EXE and PTH.DLL

- Calls `secur32.dll!LsaEnumerateLogonSessions()`
- Iterates thru sessions (LUIDs)
  - Gets username, domain, authpkg name
- Injects `pth.dll` into `LSASS.EXE`
  - Finds `msv1_0.dll!NlpGetPrimaryCredential()`
    - Not exported
    - Searches for signatures (series of fixed opcodes)
  - Calls `msv1_0.dll!NlpGetPrimaryCredential()`
    - No need to decrypt
Implementation Summary
**IAM.EXE and IAM-ALT.EXE**

- Perform ‘pass-the-hash’
- Replace current and new logon session credentials
- Two different implementations of the same ‘technique’
- IAM-ALT uses a more ‘generic’ and ‘easy’ approach and should work on more systems
- IAM uses a more ‘specialized’ approach meant to be more ‘stealthy’ (sthg like that..does not completely accomplishes this right now..)
- **WHOSTHERE.EXE** and **WHOSTHERE-ALT.EXE**
  - List credentials of current logon sessions
  - Two different implementations of the same ‘technique’
  - WHOSTHERE-ALT uses a more ‘generic’ and ‘easy’ approach and should work on more systems
  - WHOSTHERE just reads memory
    » Very safe
    » Specially to use on pentests
Using whosthere/whosthere-alt to help you “own the domain”
Compromise a Windows machine
- Dump SAM to obtain NT&NTLM hashes (e.g.:pwdump)
  » Obtains password hashes of, ONLY, users on LOCAL SAM database
How do you move from owning a single machine to owning a domain?

- Use whosthere/whosthere-alt to dump LM&NTLM credentials stored in memory
  - New logon sessions
  - Logon sessions created pre-exploitation

- You might get lucky and get accounts with **domain admin privileges**

- I’ve seen this many times.. (I’m not that lucky, so you should see the same thing 😊)

- Sometimes… logon sessions and NTLM credentials remain in memory after users log off…
DEMO
CONCLUSIONS
- PSH Toolkit brings pass-the-hash to Windows (iam/iam-alt)

- The ‘technique’ is no longer limited to certain functionality
  - You can use any microsoft and third-party tool that uses NTLM auth
  - ALL functionality of such tools is available to you
  - You can use this in a pentest (pivoting)
- Whosthere/whosthere-alt grabs hashes of (active?) logon sessions
  - Dump credentials stored in memory
  - Leave whosthere/whosthere-alt running and grab hashes of new logon sessions when they are created
  - You can obtain credentials of users not local to the workstation you are on
  - Sometimes credentials are in memory *even when users are not currently logged on*
  - Helps you own the domain after compromising only one server/workstation
QUESTIONS?
Thanks!

- Blog: hexale.blogspot.com
- My web site: www.hexale.org
- Forums: www.hexale.org/forums
**PASSTHEHASH.IDC Script**

» Finds the following Symbols
  - `?LsaEncryptMemory@@YGXPAEKH@Z`
  - `_LsapAddCredential@16`
  - `?g_Feedback@@3_KA`
  - `?g_pDESXKey@@3PAU_desxtable@@A`

» If WHOSTHERE/IAM don’t work on your system, you can make them work yourself
  » You don’t need to recompile the tools