WCE Internals

Hernan Ochoa
(hernan@ampliasecurity.com)

/Rooted CON 2011
3-4-5 Marzo 2011
Madrid

What is WCE?

- *Windows Credentials Editor* v1.0
- Manipulates *Windows Logon Sessions*
- Evolution of the Pass-the-Hash Toolkit (also written by me)
- WCE v1.1 to be published after this is over ☺

WCE features

- **Dump *in-memory* credentials of *logon sessions***
  - Lists in-memory logon sessions
    - Dumps in-memory username, domain, LM & NT hashes
    - current, future and *terminated (…)*
  - Great to 'steal' credentials not stored locally

WCE features

- **Pass-The-Hash**
  - Change/delete NTLM credentials of logon sessions
  - Create new logon sessions and associate arbitrary NTLM credentials

WCE features

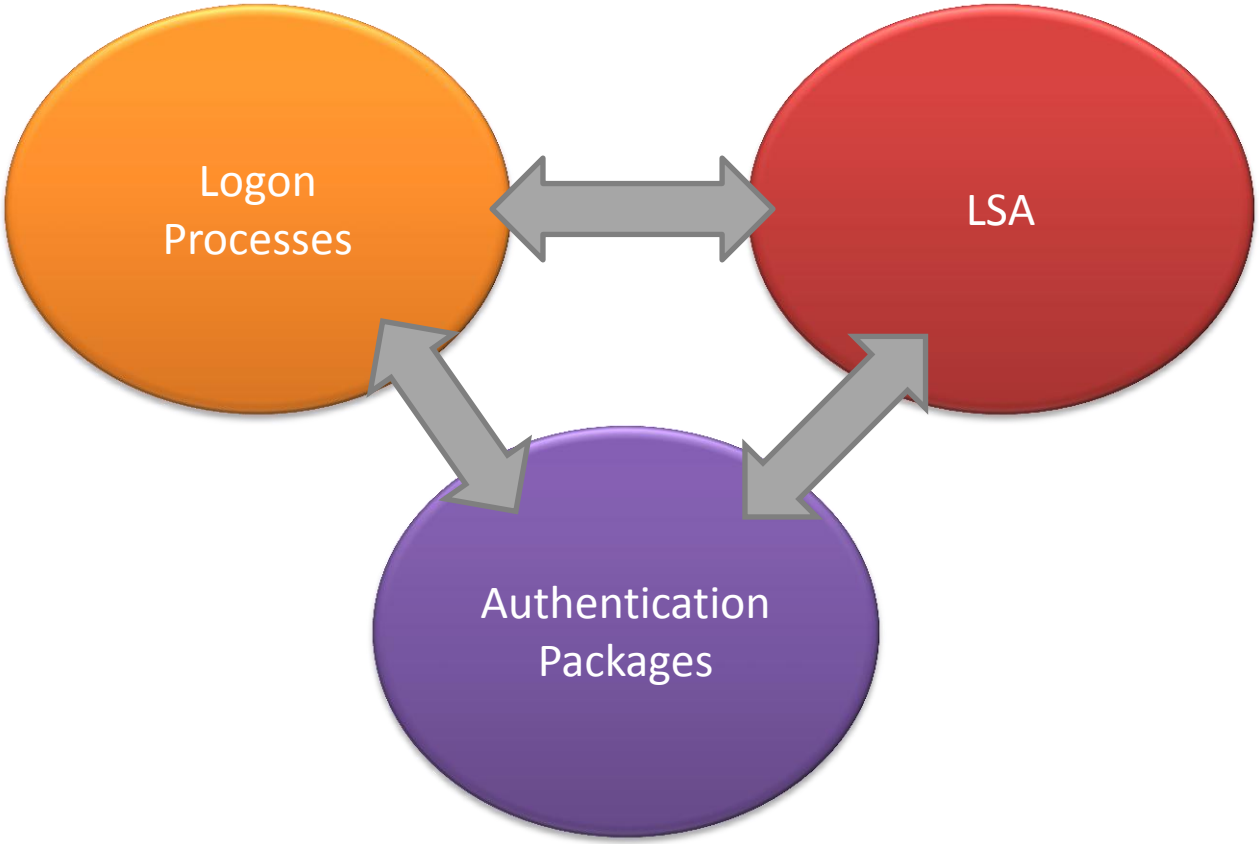
- **Does not require code injection to dump in-memory credentials (v1.1)**
  - No need to run code inside LSASS.EXE
  - Can locate, list and decrypt Logon Sessions and NTLM credentials just by reading memory

WCE features

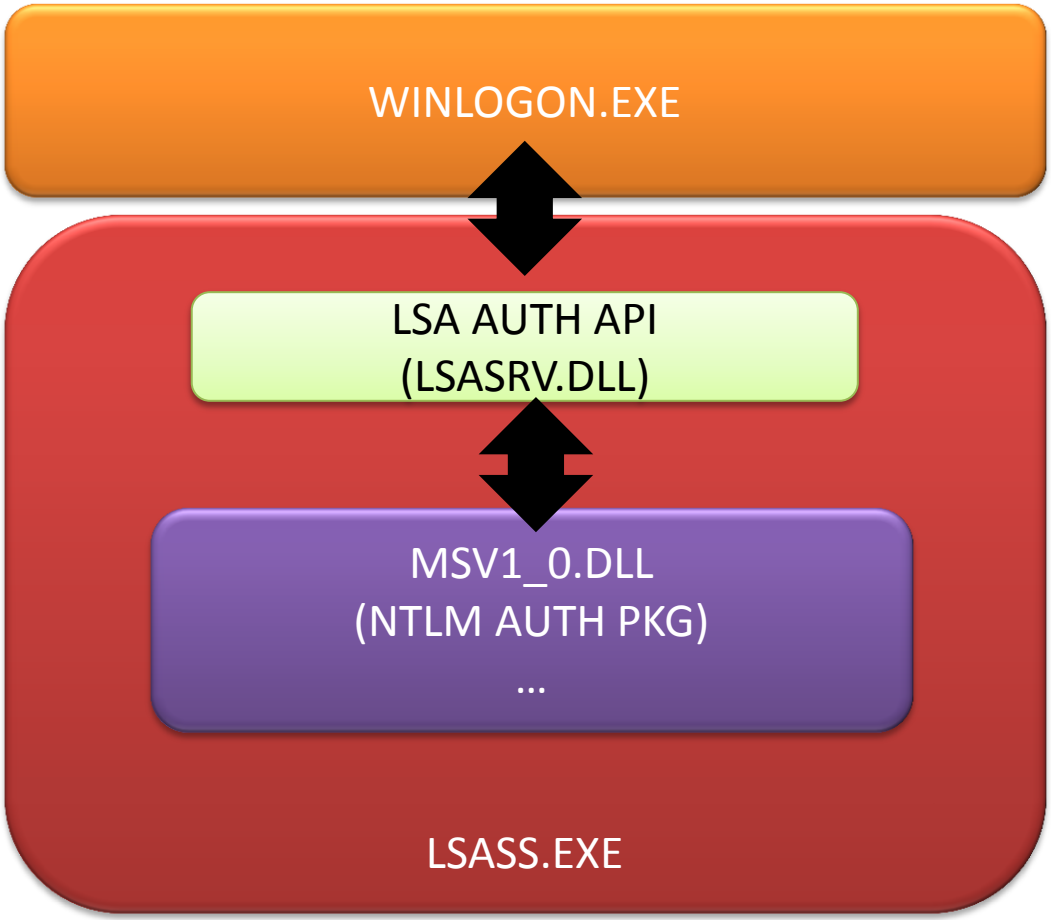
- Single executable (*wce.exe*)
  - Easier to use, upload, etc.
- Supports
  - Windows XP
  - Windows 2003
  - **Windows Vista**
  - **Windows 7**
  - **Windows 2008**

How does it work?

- Windows NT Logon and authentication model



Windows NT Logon and Authentication Model



Windows NT Logon and Authentication Model: NTLM

WINLOGON.EXE

msv1_0.dll!LsaApLogonUser/Ex/Ex2()

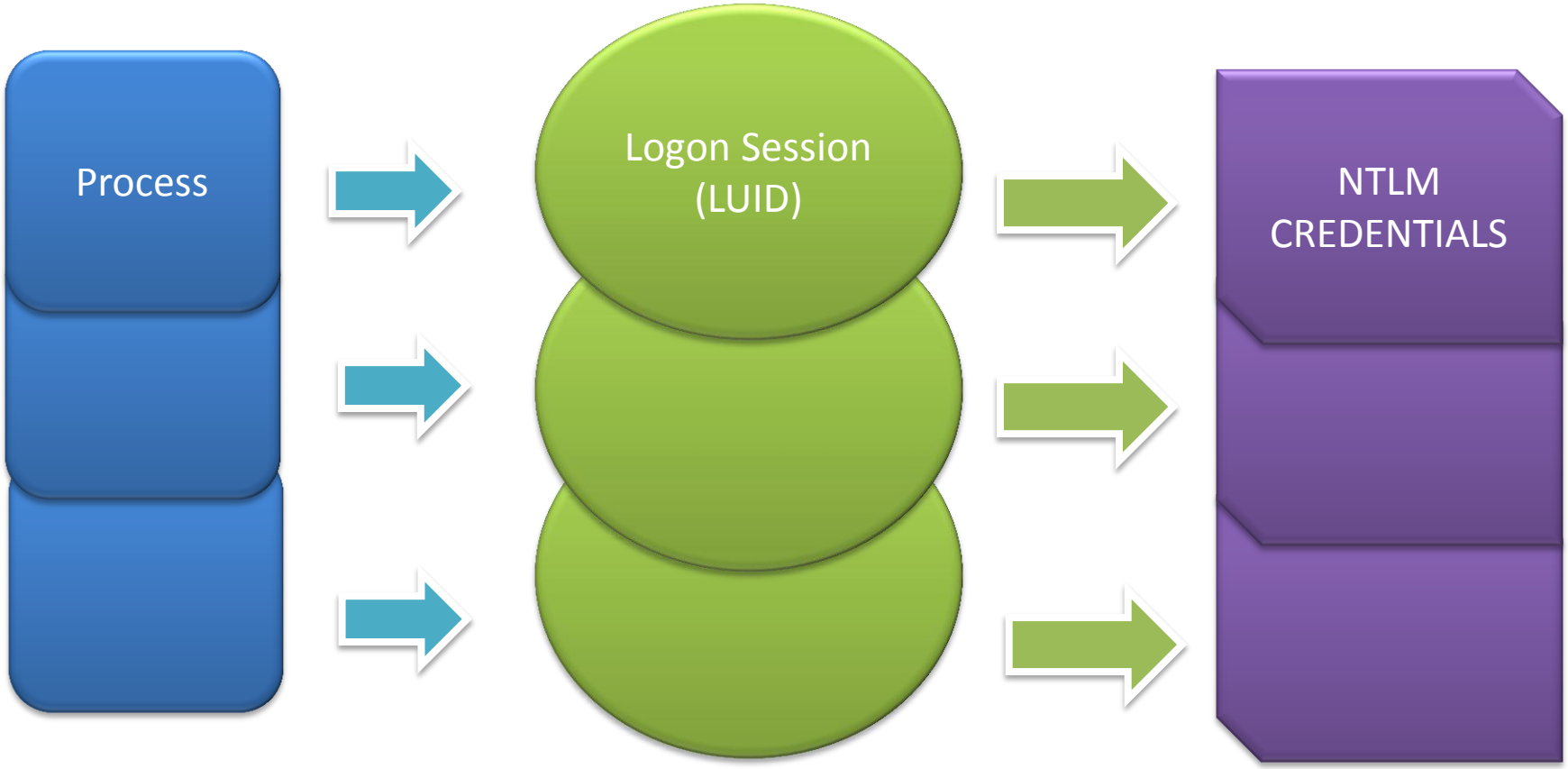
- Authenticates user
- Create logon session
- Add Credentials to Session

LSASS.EXE

NTLM CREDS

Logon Session (LUID)

Logon Sessions ⇔ Credentials



Implementation:
two possible ways…

'Use Auth Package API' Method
(less safe)

- List LUIDs
- **Run code inside LSASS.EXE**
- Call MSV1_0.DLL Functions
  - AddPrimaryCredential
  - GetPrimaryCredentials
  - DeletePrimaryCredential
- No need to encrypt or decrypt credentials
- **OS/Version ~independent**

'Read LSASS Memory' Method
(very safe)

- **Read LSASS Memory**
  - Learn inner workings
  - Undocumented structures
  - List Logon Sessions
  - Find keys and friends
  - Decrypt/Encrypt credentials
- **OS/Version dependent**

Initialization of auth packages

Loads authentication packages and calls *<authpkg>.dll!LsaApInitializePackage*

- For Example, msv1_0.dll!LsaApInitializepackage()

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

Functions provided to 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;
```

Functions handling credentials

```
NTSTATUS AddCredential(
  __in  PLUID LogonId,
  __in  ULONG AuthenticationPackage,
  __in  PLSA_STRING PrimaryKeyValue,
  __in  PLSA_STRING Credentials
);
```

Functions handling credentials

```
NTSTATUS GetCredentials(
  __in     PLUID LogonId,
  __in     ULONG AuthenticationPackage,
  __inout  PULONG QueryContext,
  __in     BOOLEAN RetrieveAllCredentials,
  __inout  PLSA_STRING PrimaryKeyValue,
  __out    PULONG PrimaryKeyLength,
  __out    PLSA_STRING Credentials
        );
```

Functions handling credentials

```
NTSTATUS DeleteCredential(
  __in  PLUID LogonId,
  __in  ULONG AuthenticationPackage,
  __in  PLSA_STRING PrimaryKeyValue
);
```

Windows NT Logon and Authentication Model: NTLM in detail

WINLOGON.EXE

LUID luid = *LsaLogonUser*(…,MSV1_0_PACKAGE_ID,…)

msv1_0.dll!LsaApLogonUser/Ex/Ex2()

- Create logon session
- Authenticates against local sam or AD
- ***msv1_0.dll!NlpAddPrimaryAddCredential***(LUID, [username, domain, LM/NT hashes],…)
  - *Lsasrv.dll!AddCredential*(LUID,…)

amplia security

'Use Auth Package API' Method

Implementation: Summary

- Find by 'signatures' and heuristics
  - *MSV1_0.DLL!NlpAddPrimaryCredential*
  - *MSV1_0.DLL!NlpDeletePrimaryCredential*
  - *MSV1_0.DLL!NlpGetPrimaryCredential*
- **Run code inside LSASS.EXE**
- Call *PrimaryCredential functions
- LSASRV.DLL functions are not called directly, eg:
  - MSV1_0.DLL!NlpAddPrimaryCredential()
    - LSASRV.DLL!AddCredential()

- **No need to encrypt/decrypt credentials**

'Use Auth Package API' Method

Implementation: Credentials Block Format

- *MSV1_0.DLL!NlpAddPrimaryCredential*(**PLUID pluid**, **BYTE* ptrtoCreds**, DWORD dwCredsSize)*;*
- *MSV1_0.DLL!NlpDeletePrimaryCredential*(**PLUID pluid**);
- *MSV1_0.DLL!NlpGetPrimaryCredential*(**PLUID pluid**, **DWORD* ptrtoCreds**, DWORD whatever);

ptrtoCreds → ?

'Use Auth Package API' Method

Implementation: Credentials Block Format

ptrtoCreds

```
typedef struct {
    UNICODE_STR ustr_domain;
    UNICODE_STR ustr_username;
    BYTE NThash[16];
    BYTE LMhash[16];
    BYTE Udomain[MAX_DOMAIN_LEN];
    BYTE Uuser[MAX_USERNAME_LEN];
} CREDSBLOCK;
```

amplia security

'Use Auth Package API' Method

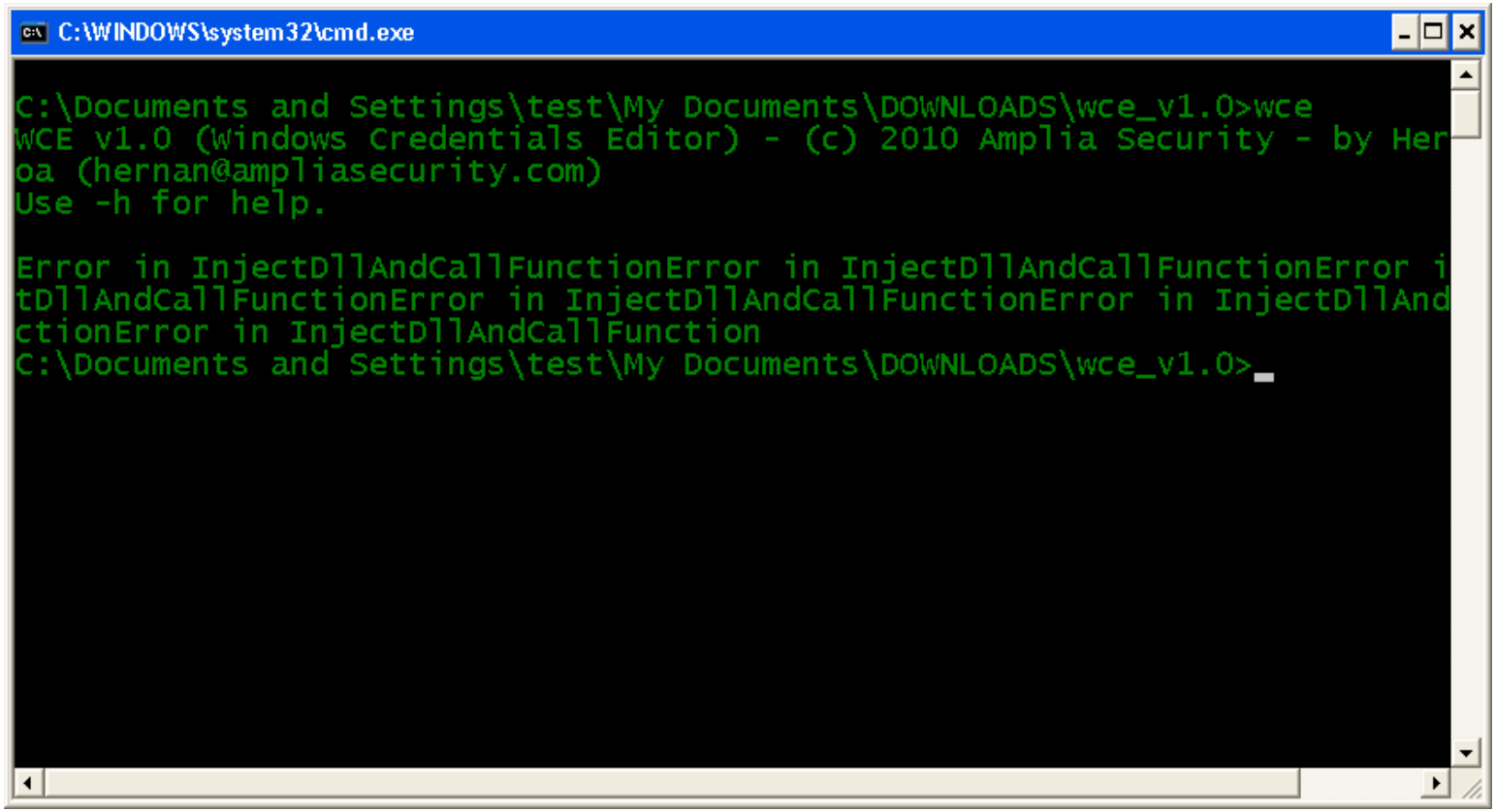
Implementation: Credentials Block Format

ptrtoCreds

+00h	000C000D
+04h	00000030
+08h	00080009
+0Ch	0000003C
+10h	11111111111111111111111111111111
+20h	22222222222222222222222222222222
+30h	D\0O\0M\0A\0I\N\0\0
+3Ch	T\0E\0S\0T\0\0

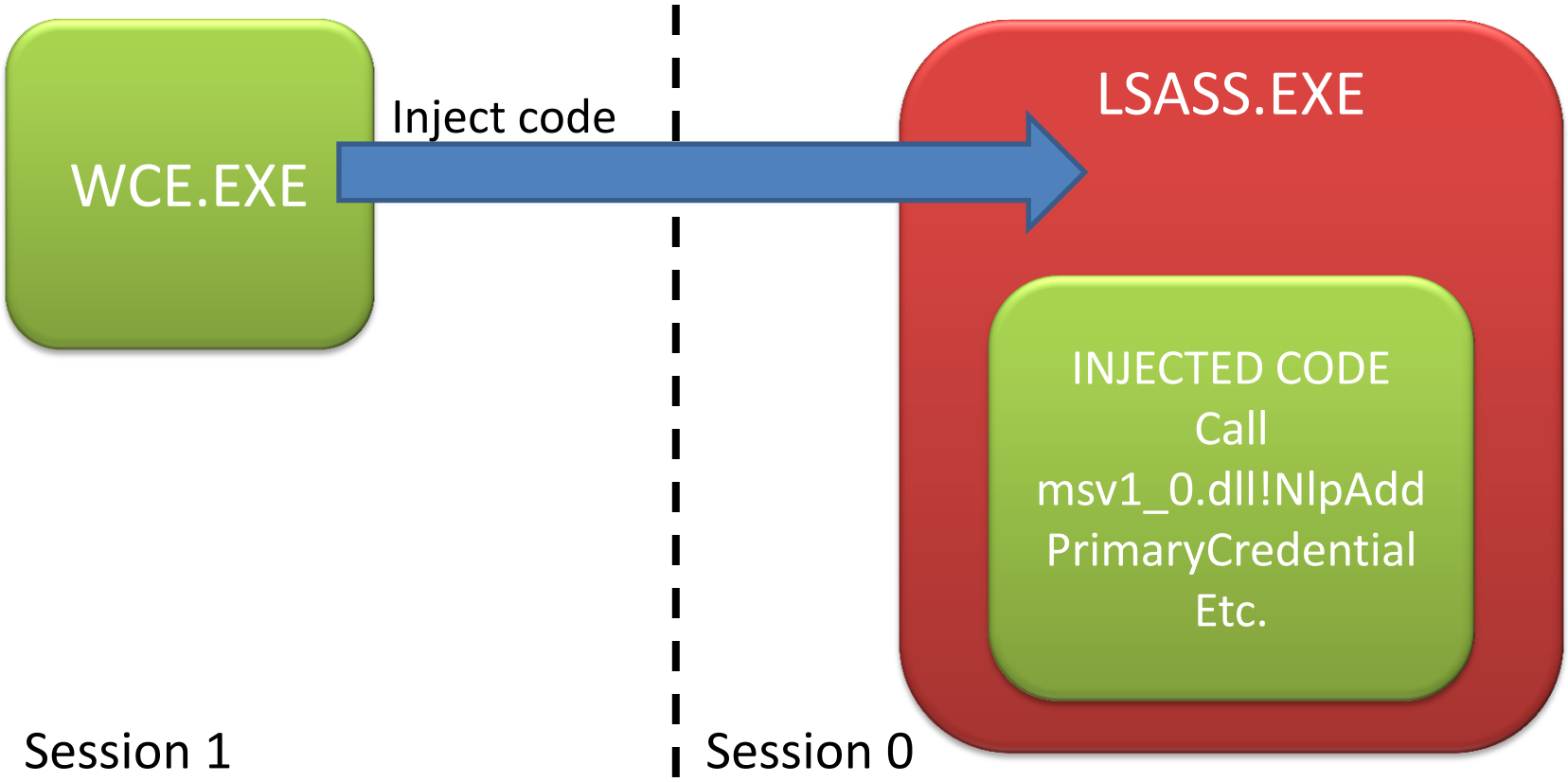
'*Use Auth Package API*' Method

Implementation:
working with Session Isolation



'*Use Auth Package API*' Method

Implementation:
working with Session Isolation



'*Use Auth Package API*' Method

Implementation:
working with Session Isolation

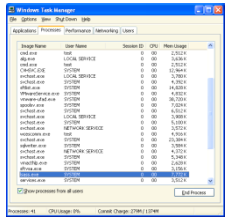


Image Name	User Name	Session ID	CPU	Mem Usage
cmd.exe	test	0	00	2,512 K
alg.exe	LOCAL SERVICE	0	00	3,636 K
cmd.exe	test	0	00	2,512 K
CVHSVC.EXE	SYSTEM	0	00	12,964 K
svchost.exe	LOCAL SERVICE	0	00	3,780 K
svchost.exe	SYSTEM	0	00	4,392 K
sftlist.exe	SYSTEM	0	00	14,828 K
svchost.exe	SYSTEM	0	00	5,348 K
vmacthlp.exe	SYSTEM	0	00	2,628 K
sftvsa.exe	SYSTEM	0	00	3,156 K
lsass.exe	SYSTEM	0	00	7,772 K
services.exe	SYSTEM	0	00	3,512 K

'*Use Auth Package API*' Method

Implementation:
working with Session Isolation

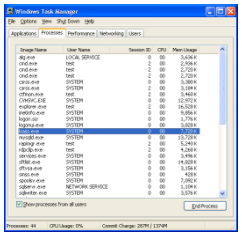
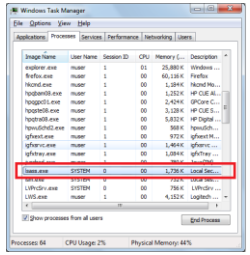


Image Name	User Name	Session ID	CPU	Mem Usage
cmd.exe	test	2	00	2,936 K
cmd.exe	test	2	00	2,720 K
cmd.exe	test	2	00	2,720 K
lsass.exe	SYSTEM	0	00	7,720 K

'*Use Auth Package API*' Method

Implementation:
working with Session Isolation

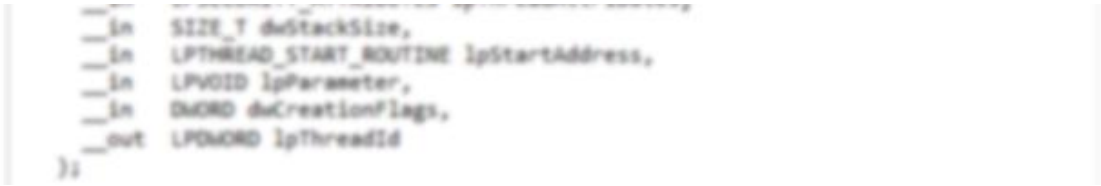


igfxext.exe	muser	1	00	972 K	igfxext M...
igfxsrvc.exe	muser	1	00	1,464 K	igfxsrvc ...
igfxtray.exe	muser	1	00	1,084 K	igfxTray ...
lsass.exe	SYSTEM	0	00	1,736 K	Local Sec...
LVPrcSrv.exe	SYSTEM	0	00	756 K	LVPrcSrv ...
LWS.exe	muser	1	00	4,152 K	Logitech ...

'*Use Auth Package API*' Method

Implementation:
working with Session Isolation

Terminal Services isolates each terminal session by design. Therefore, **CreateRemoteThread** fails if the target process is in a different session than the calling process.



In Windows Vista and later the function fails, returning NULL and the last error is being set to **ERROR_NOT_ENOUGH_MEMORY**.

(Note: CreateRemoteThread() is not the the only way to inject & run code...)

amplia security

'*Use Auth Package API*' Method

Implementation:
working with Session Isolation

- **Windows Vista/7/2008**
  - NTDLL.DLL!*NtCreateThreadEx*
- **Windows XP/2003**
  - RDP / Terminal Services
    - Create a Windows Service and do everything there
    - WCE.EXE also acts as a Windows Service
      - Installs, starts, stops and removes itself
      - IPC via Named Pipe

'Read LSASS Memory' Method

Implementation

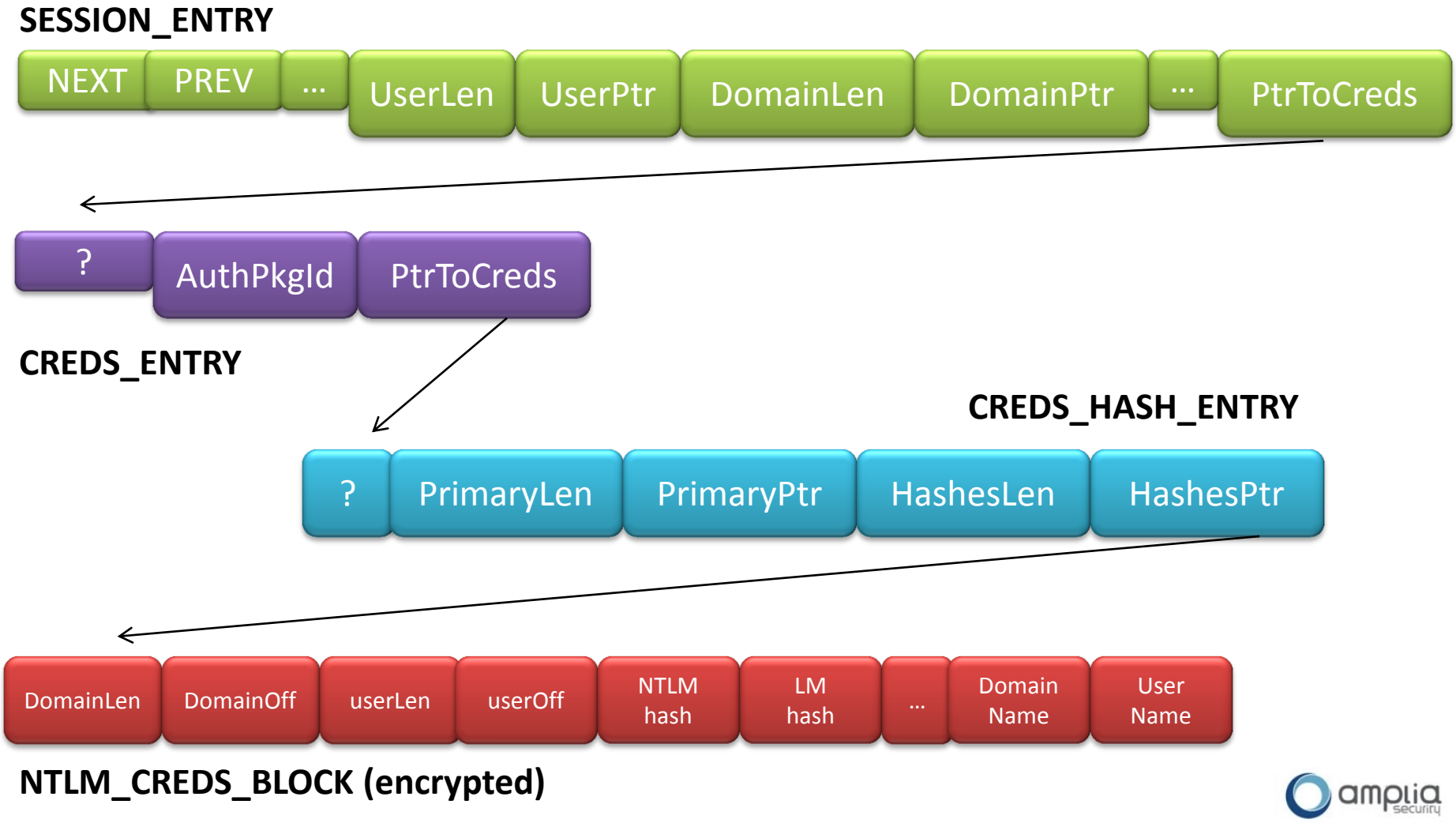
- **No need to run code inside LSASS.EXE (SUPER SAFE!)**
  - *ReadProcessMemory() only!*
- Reverse engineer inner workings of LSASS.EXE (LSASRV.DLL)
  - Structures used internally to hold logon sessions
  - Structures used internally to hold credentials
    - Structures used internally to hold NTLM Hashes
  - Decrypt credentials
    - Find keys
    - Algorithm
    - Anything else needed to decrypt (e.g.: IV)

‘Read LSASS Memory’ Method

Implementation:

Logon sessions & credentials structures

LSASRV.DLL!LogonSessionList
LSASRV.DLL!LogonSessionListCount



'Read LSASS Memory' Method

Implementation: changes in SESSION_ENTRY

Windows XP/2003

```
struct SESSION_ENTRY {
        DWORD nextEntry;
        DWORD prevEntry;
        DWORD unk1;
        DWORD unk2;
        DWORD userSize;
        DWORD userNamePtrUnicode;
        DWORD machineSize;
        DWORD machinePtrUnicode;
        ....
  +0x48 DWORD PtrToCreds;
};
```

Windows Vista/7/2008

```
struct SESSION_ENTRY {
        DWORD nextEntry;
        DWORD prevEntry;
        DWORD UNKNOWN[18];
        DWORD userSize;
        DWORD userNamePtrUnicode;
        DWORD machineSize;
        DWORD machinePtrUnicode;
        ...
  +0x88 DWORD PtrToCreds;
};
```

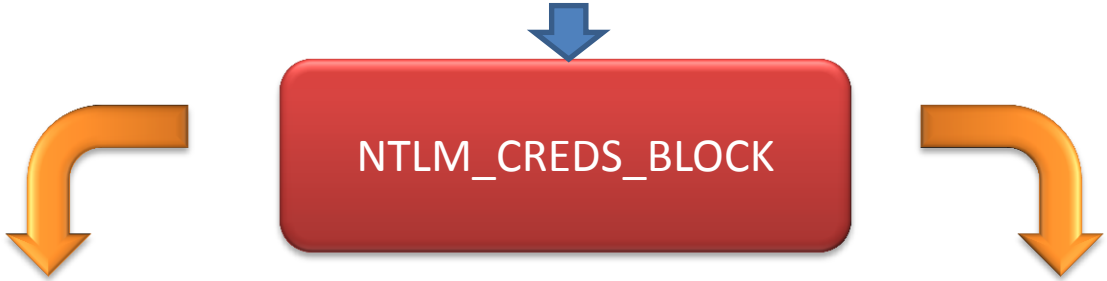
amplia security

Implementation:
LsaEncryptMemory()

Windows XP/2003

Windows Vista/7/2008

Lsasrv.dll!*LsaEncryptMemory()*



- Encrypted with **desX-CBC** or RC4
  - If mod(size/8) == 0 => desX-cbc
  - Otherwise use RC4
- Encrypted with **desX-CBC**

- Encrypted with **3DES-CBC** or AES-128-CFB
  - If mod(size/8) == 0 => 3DES-CBC
  - Otherwise use 3DES-CBC
- Encrypted with **3DES-CBC**

amplia security

Implementation

lsasrv.dll!*LsaInitializeProtectedMemory* (XP/2003)

VirtualAlloc()

↓

cbRandomKey = 100h
pDESXTable
pRandomKey

↓

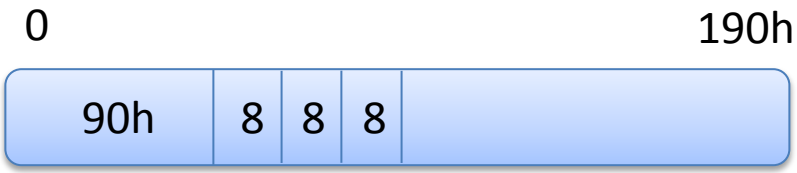
SystemFunction036(**byte Feedback[8]**,8)

↓

desxkey(pDESXTable , pRandomKey)

↓

SystemFunction036(pRandomKey, cbRandomKey)



SystemFunction036()

```
struct DESXTable  {
   byte inWhitening[8];
   byte outWhitening[8];
   DESTable desTable;
}

struct DESTable {
    unsigned long keys[16][2];
}
```

Implementation

lsasrv.dll!*LsaInitializeProtectedMemory* (Vista/7/2008)

h3DesProvider = BCryptOpenAlgorithmProvider( )
hAesProvider = BCryptOpenAlgorithmProvider( )

BCryptSetProperty(h3DesProvider, "CBCMode")
BCryptSetProperty(hAesProvider, "CFBMode")

BCryptGetProperty(h3DesProvider, "ObjectLength")
BCryptGetProperty(hAesProvider, "ObjectLength")

BCryptGenRandom(h3DesProvider, 24)
h3DesKey = BCryptGenerateSymmetricKey(h3DesProvider, 24)

BCryptGenRandom(hAesProvider, 16)
hAesKey = BCryptGenerateSymmetricKey(hAesProvider, 16)

BCryptGenRandom(**InitializationVector**, 16)

amplia security

Implementation: crypto functions used

Windows XP/2003

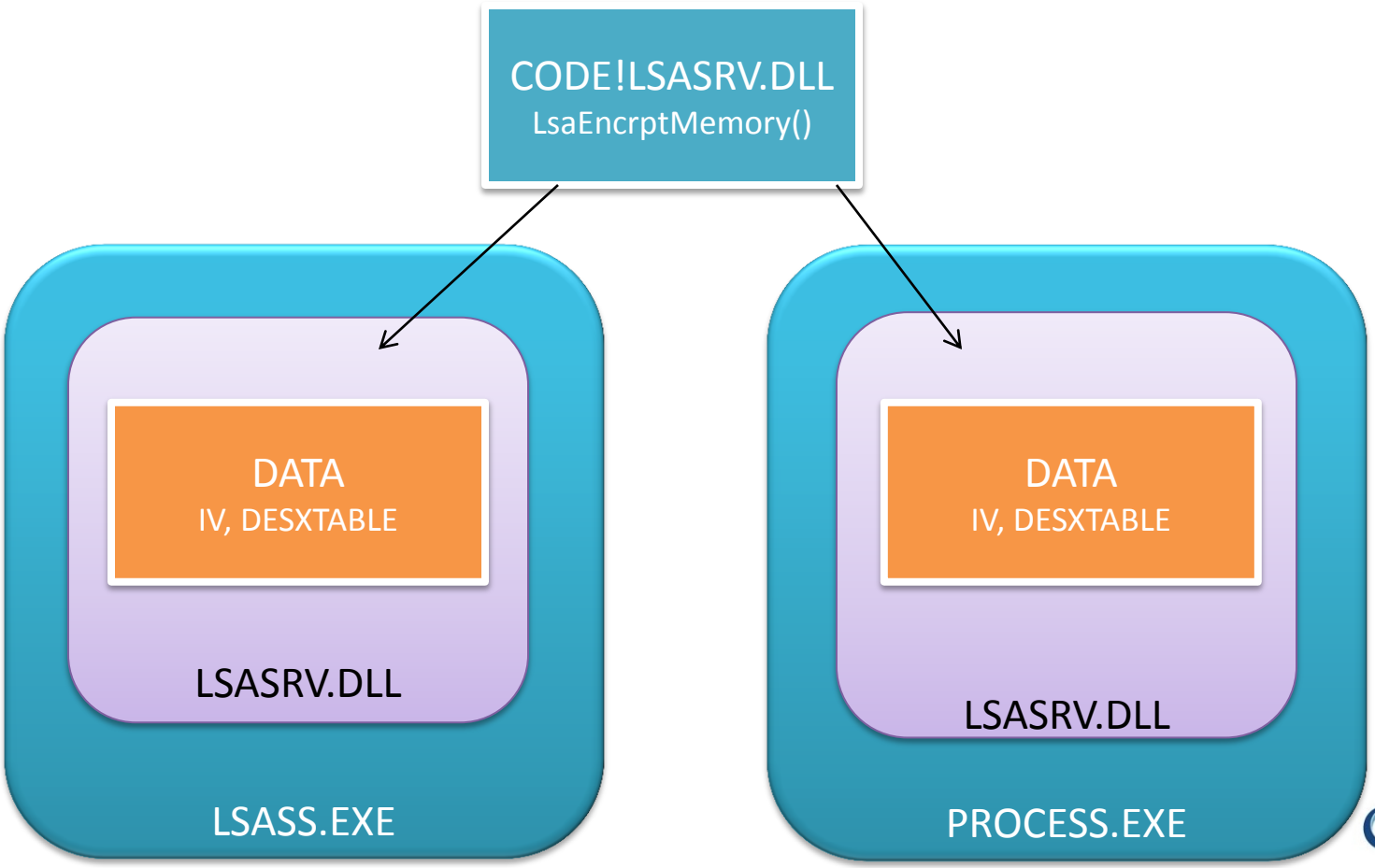
- **Uses custom desX-CBC implementation**
  - Located in LSASRV.DLL
  - Is not an API
  - Not exported by any Win32 DLL

Windows Vista/7/2008

- **Uses Cryptography API: Next Generation (CNG)**
  - Exported by BCRYPT.DLL
  - BCryptOpenAlgorithmProvider
  - BCryptSetProperty / BCryptGetProperty
  - BCryptGenRandom
  - BCryptGenerateSymmetricKey
  - BCryptEncrypt / BCryptDecrypt

Implementation

- desX-cbc 'trick' – 'Reuse' *LsaEncryptMemory*



Implementation:
pseudo-code (Vista/7/2008)

LSASRV.DLL!LsaInitializeProtectedMemory(..) {

…

h3DesKey = *BCryptGenerateSymmetricKey*(*BCryptGenRandom*(24 bytes));

…

hAesKey = *BCryptGenerateSymmetricKey*(*BCryptGenRandom(16 bytes)*)

…

IV = BCryptGenRandom(16 bytes)

}

Implementation

Finding the encryption key (Vista/7/2008)

LSASRV.DLL!LsaInitializeProtected Memory()

```
NTSTATUS WINAPI BCryptGenerateSymmetricKey(
        __inout BCRYPT_ALG_HANDLE hAlgorithm,
        __out BCRYPT_KEY_HANDLE *phKey,
        __out_opt PUCHAR pbKeyObject,
        __in ULONG cbKeyObject,
        __in PUCHAR pbSecret,
        __in ULONG cbSecret,
        __in ULONG dwFlags );
```

Implementation

Finding the encryption key (Vista/7/2008)

- BCRYPT_KEY_HANDLE hKey
  - hKey = Pointer to Memory Block (BLOB)
  - hKey + **0x3C** => encryption key

- To extract key, read from LSASS.EXE(LSASRV.DLL)
  - ((unsigned char*)h3DesKey)+0x3C
  - ((unsigned char*))hAesKey)+0x3C

Implementation

Finding the encryption key (Vista/7/2008)

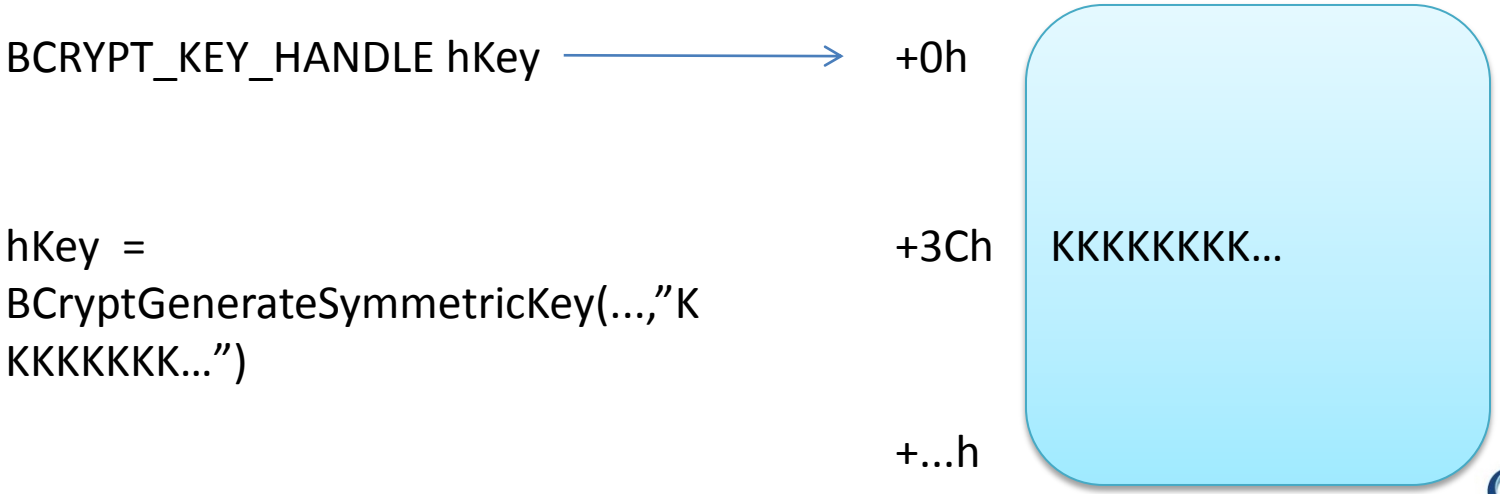
- Actually, offset changes between OSes
  - hKey + **0x3C** => encryption key (Win7)
  - hKey + **0x2C** => encryption key (Win2008)

- To be safe, I 'discover' the offset at runtime
  - I wrote a custom function for that '*KeyDiscoverOffset()*'

Implementation

Finding the encryption key (Vista/7/2008)

- *KeyDiscoverOffset*()
  - Uses CNG API to create key object with hard-coded key
  - Look for hard-coded key inside BLOB pointed to by BCRYPT_KEY_HANDLE



Implementation
Finding the IV (Vista/7/2008)

- IV is also needed
- To extract IV
  - Read IV from LSASS.EXE (LSASRV.DLL) memory
  - Symbol '*InitializationVector*'
- With IV and Key, just use CNG
  - *BCryptDecrypt* and friends
  - **No need to run code inside LSASS.EXE**

Implementation:
Addresses Needed

Windows XP/2003

- LsaLogonSessionList
- LsaLogonSessionListCount
- DESXTable
- Feedback
- LsaEncryptMemory

Windows Vista/7/2008

- LsaLogonSessionList
- LsaLogonSessionListCount
- h3DesKey
- InitializationVector

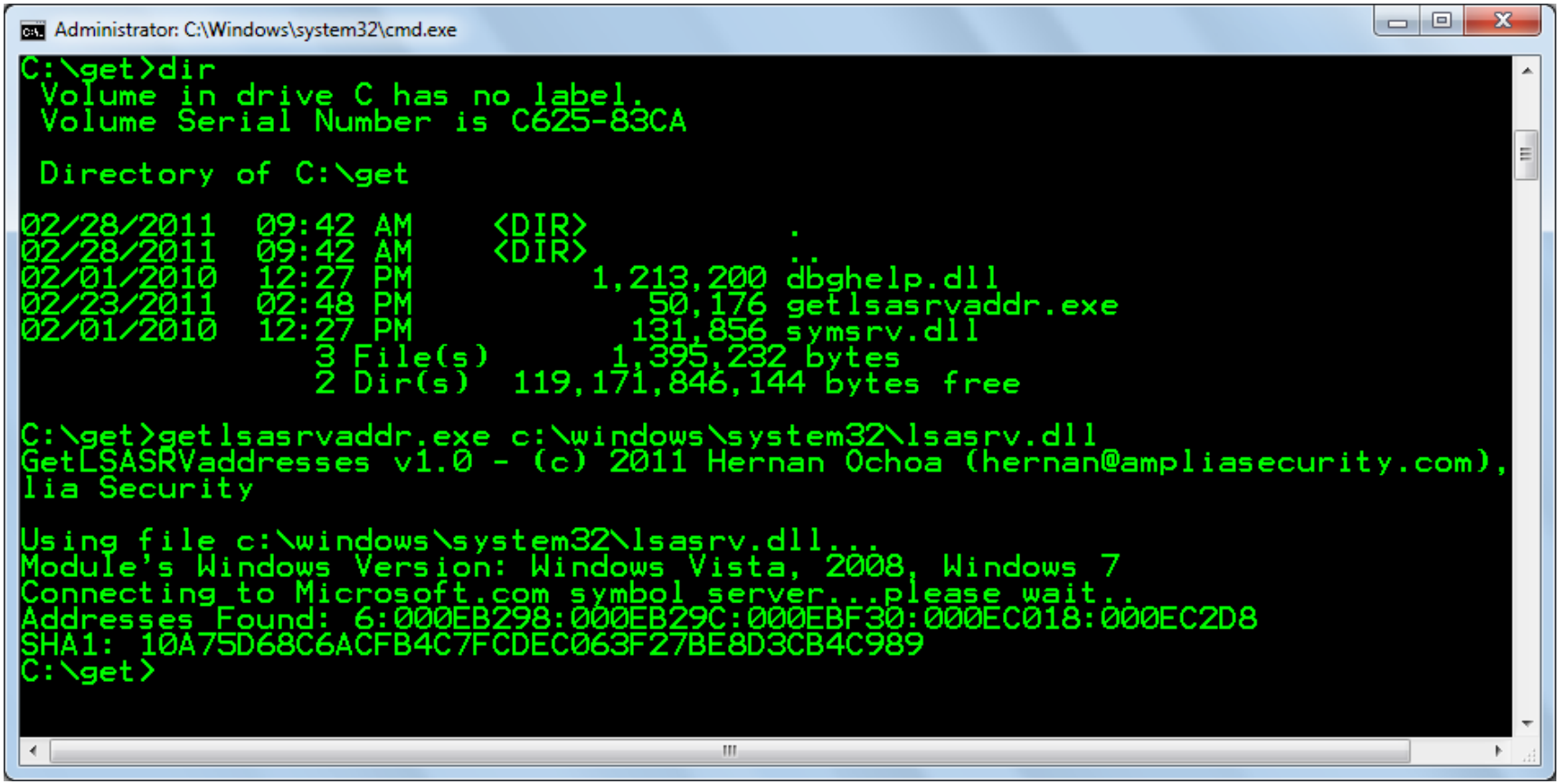
Implementation:
Addresses Needed

- Database of addresses
- ID by SHA1 hash of LSASRV.DLL
- Yes, addresses still an issue..
  - But ..
    - *Getlsasrvaddr.exe* to the rescue..

GetLSASRVADDR.exe

- Finds needed addresses automatically
  - User-friendly
  - No IDC script, IDA or anything weird like that is needed ☺
- Uses Microsoft symbol server
  - Requires http outbound connection (!)
- Associates addresses and DLLs using SHA1

GetLSASRVADDR.exe



GetLSASRVADDR.exe

- Could be integrated with WCE but..
  - The outbound connection might be an issue
  - huge not-there-by-default DLLs needed
    - *Symsrv.dll* and *dbghelp.dll* (new version, not the default one)

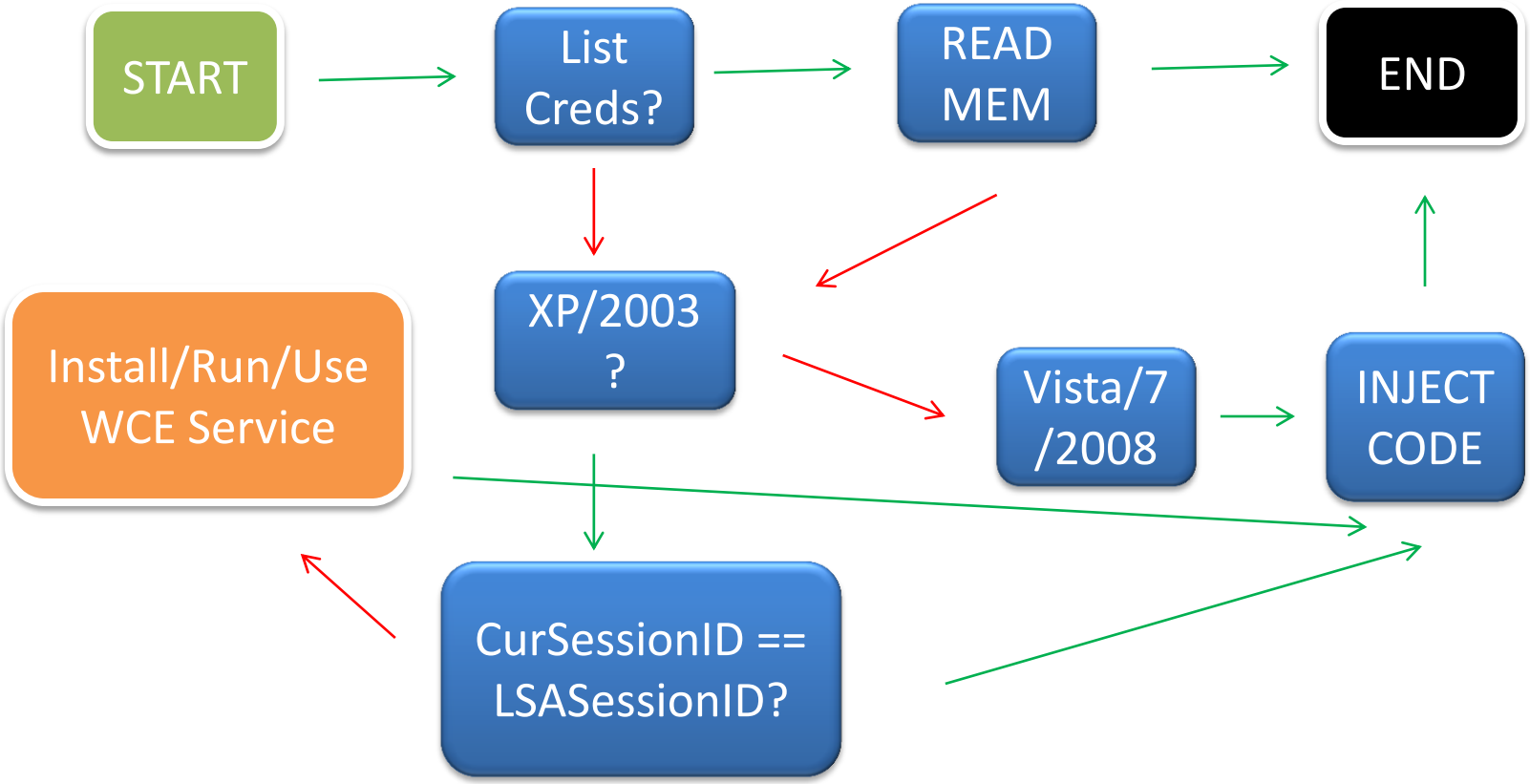
- Could implement own version of ‘symbol server’ protocol

- Or perhaps it is best to use heuristics..

Implementation:
ASLR and Windows Vista/7/2008

- LSASRV.DLL addresses and ASLR
  - Not an issue..
  - To locate symbols don't use hard-coded addresses
  - Use Offsets instead
  - ASLR is just at boot time
  - Get current LSASRV.DLL Base Address at run-time and add offset

WCE execution flow (simplified)



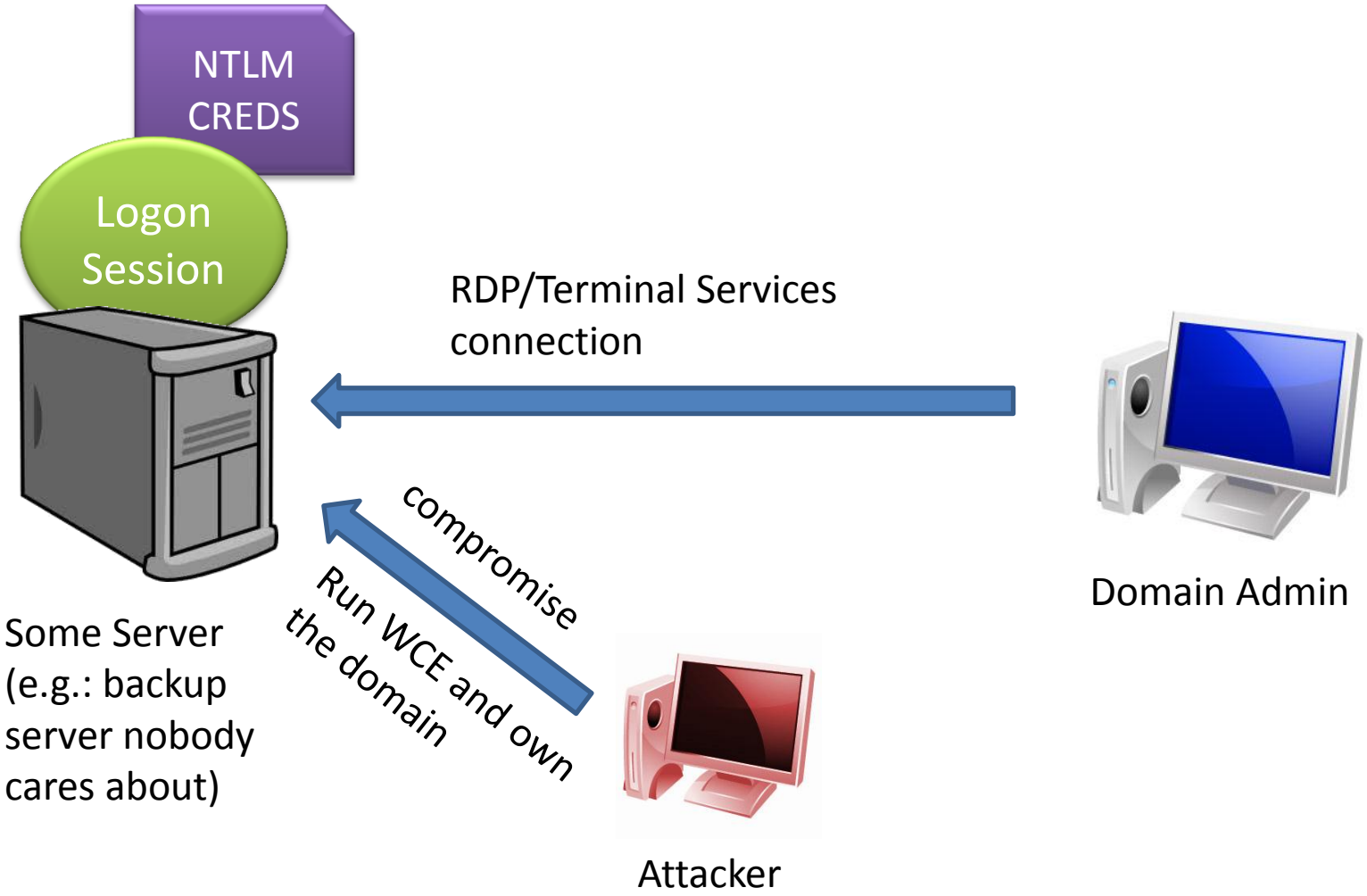
WCE vs PTH

Feature	WCE	PTH
Supports Windows Vista/7/2008	YES	NO
Single executable	YES	NO (many executables, need to upload dll, etc)
Delete NTLM Credentials	YES	NO
Works with session isolation (e.g.: via RDP)	YES	NO
Programmatic discovery of new LSASRV addresses	YES (via getlsasrvaddr)	NO
Seamlessly chooses code injection or reading from memory	YES	NO

Conclusions

- WCE v1.1
  - More features and OSes supported
  - Works via RDP/Terminal Services
  - No code injection needed
  - Better solution for ‘addresses issue’
  - ‘zombie’ logon sessions and credentials still around in Windows 7 and family..
  - Download WCE v1.1!
    - http://www.ampliasecurity.com/research/wce_v1_1.tgz

'zombie' logon sessions and credentials



Preguntas?

Gracias!

Hernan Ochoa (hernan@ampliasecurity.com)

http://www.twitter.com/hernano
http://www.twitter.com/ampliasecurity
http://www.ampliasecurity.com/blog/