Adventures in Limited User Post-Exploitation

tim elrod
and
nathan keltner

Black Hat USA 2010

Abstract

In the past year, reports have been released claiming that running as a limited or unprivileged user will mitigate up to 90% of Microsoft vulnerabilities. This claim is rooted in the belief that the potential damage that can occur via a client-side vulnerability is dramatically limited while under restricted accounts. With these articles floating in the authors' minds, they set off to determine just how much damage could still be achieved while running under limited user accounts.

This paper briefly discusses common post-exploitation activities of attackers and offers techniques for achieving similar goals under MS Windows limited user accounts, without breaking the security model through privilege escalation.

Common Post-Exploitation Techniques

Attackers are commonly interested in four primary goals: data access; using the host as a platform for attacking further systems; persistence; and covertness.

Some of the more common techniques include maintaining persistence by installing remote administration tools, rootkits, or trojan binaries; achieving covertness through log cleanup and rootkit-esque behavior; and attacking further systems through accessing stored password hashes, logging keystrokes, sniffing network traffic, and pivoting to systems accessible from the compromised system. Outside of log cleanup and rootkit-style persistence, a significant amount of the above stated behavior can be replicated under Limited User accounts.

Limited User Privileges

Without administrative access, attackers still have access to a wide variety of user level data, configuration, and system level information that will allow further control over the system. Any program can be driven on behalf of the user, allowing the set up of conditions that could allow further exploitation of the target system. Below contains a partial, non-exhaustive list of OS locations that a Limited User can access.

- File Shares and local document storage
- The HKey Current User registry hive
- The current user's configuration directories
- Much of the Windows 32 API
- Ability to modify memory space for all user-level applications
- Cache locations used by various applications, including Internet Explorer

Through these levels of access, much of the desired attack activity can still be accomplished. Specifically, through the ability to access current user registry keys, modify on-disk cache, and modify applications loaded into memory, the user's operating environment can be completely compromised, surviving reboots.

Exploiting user level privileges

1. Direct data access

It should be obvious that any data the user can access, an attacker can also access, and no additional explanation is needed. It is trivial to pull down listings of all local and network drives and files, search through these files on the compromised computer, and exfiltrate any target data observed.

2. Indirect data access, or circumventing the browser trust model

In the current world of outsourced resources, standardization, and web based applications, a significant chunk of valuable corporate transactions and data are now occurring inside the browser. As the browser runs in user space, the majority of its configuration parameters are modifiable by a limited user, and many of the files supporting modern browsers are also located in unprotected locations, the browser is the most common direct target for application interference and subversion. One particularly stealthy and persistent technique for accomplishing this attack is discussed in detail, below, in the section titled "Persistent browser cache poisoning using railgun."

3. Indirect data access cont., or a poor man's limited user sniffer

Outside of data readily accessible to users via local or remote storage, users commonly have additional access to data that is not easily observable by an attacker, and may only be present on the system while it is in transit or being viewed by the end user. For example, the authors have seen applications in use on Limited User systems actively accessing cleartext Social Security Numbers and banking data through unencrypted links to financial systems. When Administrative access is available, a sniffer can be installed that will capture the relevant traffic, achieving the attackers goal. When Administrative access is not available, however, sniffers such as Wireshark or the metasploit meterpreter payload's sniffer module cannot be employed.

All data accessible to the user is available in one form or another in memory, however, and all application interactions with the

network occur through accessible user-level code and Windows APIs. While applications cannot usually be modified on-disk, as these locations are not writable by limited users, they *are* loaded into modifiable memory where they are actually executed. By monitoring the list of running processes, in-memory attacker code can repeatedly modify application logic whenever it is seen to be running, creating the same effect as modifying the binary on disk.

Applications can be profiled for calls to common network-related Windows APIs and through the types of on-the-fly application modifications described above, the identified calls can be hooked, capturing all data sent or received from the modified application. As such, a pseudo-sniffer for specific applications can be produced, which will operate under a limited user, and that has the added benefit of being able to sidestep encryption in most implementations.

4. Password and password hash access

While gaining access to the passwords and password hashes of all cached and recent users is not possible from a limited user, it is possible to gain access to the password hash of the exploited user indirectly. While the OS disallows direct access to the user's login hashes, they are present in system level memory, and will be used by the OS to authenticate to remote resources when instructed to by user-level code.

One example for exploitation in an Internet facing attack, where an internal client has been compromised and only Limited User access has been obtained, is to fool Internet Explorer into authenticating to a server under attacker control. IE, by default, will utilize Integrated Windows Authentication over HTTP/S to any sites determined to fall under the Intranet security zone. Adding sites to the Intranet zone is as simple as modifying key registry entries under HKCU. This hash can be replayed in a live attack against additional resources, such as externally available sites or services that utilize NTLM and Active Directory integration. In addition, when an XP client has been compromised, default behavior allows for retrieval of hashes susceptible to trivial hash cracking of the first 7 characters of the password, utilizing rainbow tables.

5. Keylogging

In various discussions with security professionals around the country, confusion also appears in the understanding of the access rights required to perform keylogging functionality. User level keylogging is trivial with, for example, the metasploit meterpreter payload's keylogger extension. While keys pressed by additional users outside of the current user's session are unavailable, all keystrokes from the compromised session are readily accessible.

6. Persistence

With only user level privileges, maintaining access and persistence has been done for many years by malware. While not a new technique to anyone paying attention, security professionals, particularly those playing defense, seem to not be aware of its usage in non-Administrative environments. Specifically, from the use of the startup folder in the start menu to the run* keys in HKCU\software\microsoft\windows\CurrentVersion\ registry keys, ensuring application code loads on boot is trivial. Once code is running, it can begin to circumvent any other code currently running in user space through any of the aforementioned means.

Admin access may be a hindrance

In high security environments observed by the authors, accounts with Administrative level access are often closely monitored and access to sensitive data is commonly restricted to only Limited Users with a legitimate business need for access. In these scenarios, accessing target data under an Administrative account often requires modifying file, directory, or database permissions to allow access to the account in use. Both the access modification and the act of an Administrative user viewing data normally only viewed by known, Limited Users, are flags for security team follow ups and have led to discovery of compromise.

Often in an effort to maintain a level of stealth, the attacker will try to compromise user level accounts that can access sensitive information in a way that will not raise the suspicions of administrators or security professionals. For this reason, administrative access is not always needed or desired for accessing the types of highly sensitive information targeted by criminals and penetration testers alike.

Persistent browser cache poisoning using railgun

Browsers rely on local caches of web content to minimize the amount of data required to traverse the network and speed up content access to end users. Site operators have the option of sending down cache headers in HTTP responses, instructing the browser to cache the relevant file for a site-configured period of time. When instructed by the site, the browser writes this code to cache directories on the local disk and updates a database containing metadata about the cached code, such as received headers, date and time received, how long to cache the file, etc. Once cached, the browser will pull code from this location on future visits, rather than pulling code over the network from the accessed site, until such time that the cache expires.

During talks entitled "Wireless security isn't dead, attacking clients with MSF" and "Wifi Security -or- Descending Into Depression and Drink" at BlackHat DC 2010 and ShmooCon 2010, respectively, Mike Kershaw (aka Dragorn) described an attack against this functionality via unencrypted wireless links. With header and content control, arbitrary cache entries with attacker determined expiration entries can be created.

Through code execution loaded from the cache, attackers have full control over any logic and data passing through the targeted site within the browser. For example, javascript can be inserted to modify all HTTPS form posts to submit data to both an attacker controlled server, as well as the legitimate site. If done correctly, it would be unobservable to the victim.

Additionally, no external code need be resident on the system after this attack has taken place, and even if the compromise/attack were detected, the maliciously cached entries will persist until the browser cache is manually cleared or the OS is reinstalled. In most organizations observed by the authors, systems are 'cleaned' of malware rather than wiped, which would result in continued compromise in user data from this attack.

A similar attack utilizing Win32 function calls to force caching of malicious code directly, with arbitrarily long cache expirations, is proposed below.

Specifically, wininet.dll offers the API for applications to interact with windows internet functionality. This is also the DLL responsible for reading and writing to the Internet Explorer cache.

To poison this cache, two functions in wininet are used. The first, CreateUrlCacheEntryA, will cause creation of a randomly named file in the cache directory for a given URL, and pass a file handle back allowing read and write access. After writing to this file any malicious content desired, this entry is committed to the IE cache by calling CommitUrlCacheEntryA. This function will commit the entry's metadata into the IE cache database. If a cache entry exists for the URL being committed, Windows will overwrite it with the malicious cache entry.

Following are the function prototypes and descriptions for the two functions used in browser cache poisoning.

```
BOOLAPI CreateUrlCacheEntry(
 __in  LPCTSTR lpszUrlName,
 __in  DWORD dwExpectedFileSize,
 __in  LPCTSTR lpszFileExtension,
 __out  LPTSTR lpszFileName,
 __reserved DWORD dwReserved
);
```

The CreateUrlCacheEntry function is fairly simple, requiring the following arguments: a string for the URL being cached, the expected file size (passing null will cause this entry to be variable), the file extension for the cache file, and the size of the path returned back as lpszFileName (normally this is set to MAX_PATH+1). If successful, the function will return true and output a file path to the created cache file.

```
BOOLAPI CommitUrlCacheEntryA(
 __in  LPCSTR lpszUrlName,
 __in  LPCSTR lpszLocalFileName,
 __in  FILETIME ExpireTime,
 __in  FILETIME LastModifiedTime,
 __in  DWORD CacheEntryType,
 __in  LPBYTE lpHeaderInfo,
 __in  DWORD cchHeaderInfo,
 __reserved LPCSTR lpszFileExtension,
 __in  LPCSTR lpszOriginalUrl
);
```

The CommitUrlCacheEntryA function, similarly, requires the following arguments: the URL to cache, the local file name containing the cache data (this is the temp file returned from CreateUrlCacheEntryA), the expiry time and last modified times (passed in as null and later controlled through HTTP headers), and the cache entry type (a set of constants used to express what type

of cache entry this is). By default, a normal cache entry will be scavenged in 10 minutes after being written to disk, but by passing the STICKY_CACHE_ENTRY constant it is possible to survive this scavenging process. After this, the HTTP headers for the entry are included, and by using the cache-control http header arbitrary expiration dates can be achieved, allowing entries to poisoned indefinitely.

With Internet Explorer 8, there are two types of IE caches: the normal content.ie5 cache that hasn't changed since Windows 2000, and the new low privilege /low/content.ie5 directory. By default, Internet Explorer 8 will use the low privilege cache. To write to this newer cache directory, the caller of the function must have its Integrity Level set to low, through migration into an appropriate process or through the cli command "icacls program.exe /setintegritylevel low". With the appropriate Integrity Level, the functions can now be used to write to the low/conent.ie5 cache.

Alternatively, by putting the target URL in the trusted sites list, Internet Explorer 8 can be forced to use the normal content.ie5 cache.

Both of these techniques may be useful in different situations, and the implementation being released supports both of these methods for poisoning the Internet Explorer 8 cache.

In summary, through these techniques it is possible to poison content loaded into the DOM for targeted, sensitive websites, capturing authentication information and data for sites like webmail, backend office applications, and SSL VPNs. Additionally, it does so without leaving any easily identifiable code resident on the victim's system, and will persist until the cache entry expires, the cache is manually wiped, or the user's system is rebuilt.

In July, 2010, the Metasploit Project integrated a new meterpreter extension, railgun, into its development tree. This extension allows scriptable access to the target's Win32 API, including the loading of arbitrary DLLs into memory and the calling of arbitrary functions. This tool was extended to allow for the calling of the above wininet.dll functions. In conjunction with the presentation accompanying this paper, scripts utilizing railgun functionality to implement the cache attacks outlined herein are being released.

Conclusion

Running as a limited user is a worthwhile security practice and should be utilized as a defensive technique available to those protecting corporate and national infrastructure. However, its context and efficacy must be properly understood, including impact on Incident Response planning and overall security architecture.

For an updated version of this paper, including code snippets, typo fixes, flying rainbow sheep, and a full reference listing, please visit:

http://www.ri0tnet.net/LimitedUserPostEx/