

Insecurities within automatic update systems v1.16

Can patching let a cracker in?

ing. P. Ruissen
ing. R. Vloothuis

Research project 2
MSc in System and Network Engineering



University of Amsterdam
Class of 2006-2007

July 1, 2007

Contents

Abstract	3
Preface	4
1 Introduction	4
1.1 Hypothesis	4
1.2 Related work and literature	4
1.3 Scope, deliverables and planning	4
2 Abstract model for automatic update systems	5
2.1 Introduction	5
2.1.1 CIA triad	5
2.2 Confidentiality	6
2.2.1 Introduction	6
2.2.2 Theoretical solutions	6
2.2.3 Implementation	6
2.2.4 Conclusion	6
2.3 Integrity	7
2.3.1 Introduction	7
2.3.2 Theoretical solutions	7
2.3.3 Implementation	8
2.4 Availability	9
2.4.1 Introduction	9
2.4.2 Theoretical solutions	9
2.5 Overall Conclusion	9
3 Method of analysis	10
3.1 Summary of attacks	11
3.1.1 Confidentiality	11
3.1.2 Integrity	12
3.1.3 Availability	12
3.2 Consequences	13
4 Attacking in the field	14
4.1 Introduction and tools	14
4.2 Operating systems	15
4.2.1 Windows XP	15
4.2.2 Windows Vista	17
4.2.3 Redhat package manager	18
4.2.4 Debian Advanced Packaging Tool	19
4.2.5 Gentoo Portage Tree	20
4.3 Applications	21
4.3.1 Mozilla Firefox and third party plugins	21
4.3.2 Java virtual machine	23
4.3.3 Adobe acrobat reader	25
4.3.4 Command & Conquer 3, Electronic arts	27
5 Risk Assessment	29
6 Conclusions	31
6.1 Recommendations	31
6.1.1 Developers	31
6.1.2 Users	31
6.2 Future work	31
References	32
Appendix	34

Abstract

This document describes the risks of update mechanisms. An abstract model is described which explains the inner workings of an ideally update mechanism which can be implemented in applications and operating systems. The CIA¹ triad model is utilized in order to establish security. With this model, an attack is launched on a selection of applications and operating systems. During our research, we found some applications did have security flaws in their update mechanisms. These flaws can be exploited and undermine the integrity of a system which has this software installed. Therefore these applications were labeled a higher risk in the risk analysis than others. We conclude update mechanisms have improved over the last few years, but a single faulty mechanism can still compromise the entire system.

Preface

This report is written for Research project 2 as part of our Master of Science study in the field of System and Network Engineering at the University of Amsterdam. This research was performed on behalf of KPMG EDP Auditors N.V, Amstelveen under supervision of ing. Jeroen van Beek MSc CISSP², drs. J.G. Hans Ijkel CISSP from IT Advisory. We want to thank them for their enthusiasm and excellent guidance during this project. Further, we want to thank KPMG and dr.ir. C.Th.A.M. de Laat from the Faculty of Science for making this research possible.

This document describes research in the field of security auditing, evaluating the risks of update mechanisms. The first chapter describes the importance of this project, the hypothesis and related work. The second chapter describes an abstract model with the requirements necessary to establish a secure update mechanism. Chapter 3 describes our methods of analysis and chapter 4 will discuss the results of our tests on well known update mechanisms. Chapter 5 will discuss the results, risk analysis, recommendations and conclusions.

¹Confidentiality, Integrity and Availability

²Certified Information Systems Security Professional

1 Introduction

Patch management used to be a difficult and error prone task. Patches were distributed on storage devices and manually applied on each machine. These routines made patching a slow and risky process. Wrongly applied patches cause mean failure of one of the core processes within an organization. But with the rise of the software industry this all changed. Developers started releasing software packages on public mirrors to distribute patches to all customers. Patches are automatically applied by update systems which have the ability to periodically check, retrieve and install packages without user interaction. Almost all important widely used application or operating system incorporates an update mechanism.

1.1 Hypothesis

Automatic update mechanisms despite their seeming convenience can also introduce security risks. If the update mechanism can be influenced in some way, malicious code might be executed on a system. It is easy to imagine what kind of consequences this might have to this system and maybe even the entire network. For instance, unauthorized access could be obtained through a faulty update mechanism and company data might be stolen. Therefore, keeping a system secure is vital, thus designing such a mechanism should be done careful.

In this research project we look at the security of these update mechanisms.

This leads to our main research question:

Which requirements are necessary for an update mechanism in order to operate safely and in which degree are current operating systems and applications vulnerable for update mechanism attacks?

We divide the research in two parts:

- abstract model which describes how a security update mechanism should work.
- testing several commonly used applications and operating systems to see if the implementations adhere to the items as documented in our abstract model

1.2 Related work and literature

Little research has been done on automatic update insecurities. The difficulties with regard to automatic update insecurities have however not been widely accepted and picked up. Exceptions to this are several authors [1, 2, 3, 4] who documented currently known problems in update mechanisms. These authors mainly focus on insecurities within applications such as McAfee virusscan, which for instance contained weaknesses in its update mechanism and others also proved to be vulnerable. The author of article [1] pointed out that a significant amount of vendors have made progress in improving update mechanisms. McAfee virusscan for example, has included authenticated binaries [1] which greatly improved the security of the update mechanism. Nevertheless, the existing research does not adequately address important issues such as requirement to safely applying security updates and risk assessments.

1.3 Scope, deliverables and planning

For this research, we have applied the following research plan:

- Week 1** Literature review, define operating systems and applications to be tested.
- Week 2** Abstract model, requirements and security of update mechanisms.
- Week 3** Practical work, test models and vulnerability testing.
- Week 4** Risk analysis based on our findings, feedback report and presentation.

In order to meet the time schedule, we set some boundaries for this project by reviewing popular applications and operating systems:

Applications:	Mozilla Firefox 2.0	Operating systems:	Windows Vista
	Adobe Acrobat reader 8		Windows XP SP2
	Java Virtual Machine 1.5		CentOS 5 Yum (Redhat based)
	Command & Conquer 3		Gentoo Portage
			Debian APT 0.6

2 Abstract model for automatic update systems

2.1 Introduction

As part of the application security, an update mechanism must also be secure enough to ensure safe patching. Designing an update mechanism which is secure can be a difficult task. Therefore, it is important each designed mechanism is confronted with a model to evaluate the security. There are certain quality demands which must be met. According to ISO 9126-1 on software quality, all software including update systems are bound to certain quality demands. The ISO model describes several characteristics which can be used to ensure software quality. We can summarize these items into the following criteria:

Functionality requires that software operates according to the implied needs. Update mechanisms in general must be accurate, interoperate with other software and compliant to standards.

Reliability The process of updating must be reliable and fault-tolerant. If patching has certain side effects, the administrator must be able to fallback to a previous situation without affecting other applications or performance. The development of update packages should be accompanied by thorough testing as they can have a great impact on a system.

Usability Applying an update should be easy for normal users in a time efficient manner. Farms of repository servers must easily be administered. The release cycle of security updates should be short.

Efficiency handles the performance under different conditions and effective use of time and system resources.

Maintainability describes the needs for changeability, testability and stability. Automated package generation and tight version and access control must be integrated and collaborate with package management systems.

Portability requires easy adaptation of software and migration to other environments.

2.1.1 CIA triad

For this project, we mainly focus on the security aspect which is a sub-characteristic of functionality in the ISO model. To ensure security we identify three demands which are also known as the widely known CIA triad: Integrity, Confidentiality and Availability.

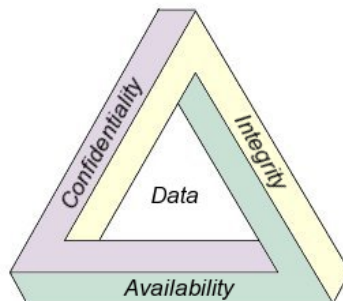


Figure 1: CIA triad

In order to translate this model to the security update mechanism, the following model is established:

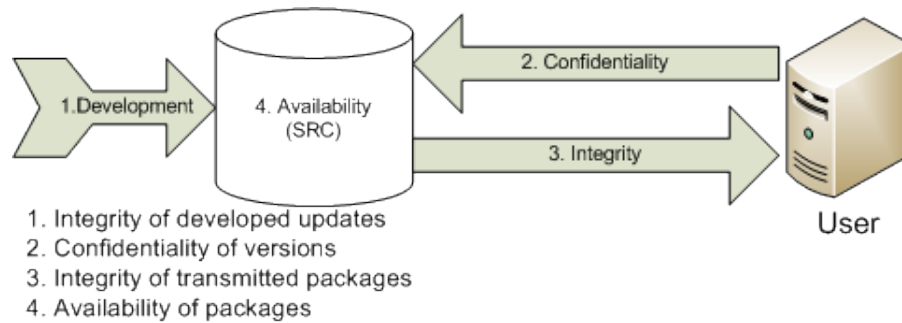


Figure 2: Retrieving updates from a repository

2.2 Confidentiality

2.2.1 Introduction

Confidentiality is an important criterion for security. It is of importance that the provider and the user are the only parties involved in the update procedure. It is considered to be a threat when confidentiality is compromised, first because a possible attacker might capture an update transfer and discover the internal workings of the update mechanism. This concept is known as sniffing. By keeping confidentiality you can prevent eavesdropping.

Another reason why a user wouldn't want anyone to monitor the update process is privacy. Privacy sensitive information like application types, versions and other plain data should not be known to a possible attacker. This knowledge can be abused in case of a vulnerability in a specific version of a program.

2.2.2 Theoretical solutions

A typical solution to ensure confidentiality and privacy is to encrypt transmitted data. Users want to make sure the transfer of the update package is done encrypted and no one is able to read its contents.

2.2.3 Implementation

A widely known implementation in preventing sniffing and data tampering and provide server authentication is to use SSL version 3 (Netscape). The most recent version of this approach is TLS version 1 (RFC 2246) released by the IETF. SSL and TLS are encrypted protocols to provide secure communication. The transactions are asymmetric, leaving the client anonymous. TLS is deployed by many leading financial institutions who consider security as their top priority and proven to be secure.

2.2.4 Conclusion

The best way to ensure confidentiality is by using TSL encryption.

2.3 Integrity

2.3.1 Introduction

For this research, we understand by integrity[32]:

reassurance to the recipient that the message has not been altered since it was transmitted by the sender.

Integrity refers to the validity of data. Validity might be impaired by transmission faults and alteration of the contents by attackers.

Transmission faults The underlying data and network layer protocols provide limited protection of transmission. They for instance apply handshaking and error correction. However, package corruption might still occur and might have impact on the target environment or application itself. Damaged or corrupted binary update packages might not be applied because it cannot be decompressed or executed. This can cause unpredictable situations.

Unauthorized alteration of contents Sources of update packages should also be checked if they contain valid data. Altering a source and its packages by an attacker, can render a system vulnerable if there is not an integrity check of the source in place. Developers should be prevented from injecting malicious code into update packages. A corrupt developer with bad intentions becomes a threat when he can inject code without it being checked. Another dangerous risk is man-in-the-middle attacks which are described in chapter 3. Man-in-the-middle attacks can be used to alter the contents of an update package during transmission.

2.3.2 Theoretical solutions

To address these potential risks, the following solutions can be suggested.

Checksums is a form of error detection or correction (redundancy check) to prevent corruption of the original message. Parity checking is the simplest redundancy check available, but there are more efficient algorithms like CRC that not only store the sum of bytes, but also the position in the data stream. Cyclic redundancy checks (CRC) are primarily used to ensure a file is transferred intact. The algorithm takes a string or a file as input and produces a fixed length output value. CRC only ensures error correction and detection and prevents data corruption. CRC cannot be used as a security measure because the algorithm is easily circumvented.

Hashing algorithms A hash is a digital fingerprint or hash sum of a file, which is derived from a one way mathematical algorithm. From this hashing process, a sum is produced in a fixed length depending on the algorithm used. The better the hash algorithm, the more difficult it is to find a second file, which produces the exact same sum. Hashing or so called message digest only provides error detection and integrity of the files. Hashing alone does not provide authenticity.

Digital signatures Digital signatures must be used to authenticate the source of any update package. These are created by an asymmetric public key pair and a hash algorithm like SHA2 (RFC 4634) or MD5 (RFC 1321) to hash the package and create a message digest. The message digest can only be used to check the integrity of the package and prevents data corruption. A digital signature is made by encrypting the message digest with a private key and appending the signature to the package.

Web of trust Decryption using the public key and comparing hashes does not necessarily authenticate since both keys and the digest can be generated by a malicious attacker. For this reason it is necessary that a third party like a group of colleagues or a certificate authority center signs the public key. The key becomes a certificate when it contains information about its source and is signed by a trusted third party (Verisign, CAcert). The role of a trusted third party is usually the company in case of proprietary software.

2.3.3 Implementation

Theoretical concepts named in the previous paragraph maybe theoretically sound, but in most cases present flaws in wrongful implementations.

Auditing packages Before a package reaches the stable tree it should be well audited on security flaws. Thoroughly testing is imperative, since even one line of code can make a system vulnerable. Especially in a distributed environment where every potential developer can contribute source code, reviewing, testing and qualifying packages on payload very important. Although part of these security assessments can be done automatically, some Trojans and viruses can only be found by manually reviewing the code. We recommend updating packages from trusted mirrors, because installation often need root privileges. Packages from untrusted mirrors with or without self signed certificates might compromise the local system.

Public key infrastructures The best way to secure an update session is to use a form of public key infrastructure (PKI). This can be a form of PGP (RFC 1991) (Pretty good privacy) that relies on signed public keys. Digital (public) certificates must be signed by a independent and trusted third party to ensure validity. Therefore, most organizations use a release-signing key to sign every release file of official packages. From there, an encrypted and safe update session can be performed automatically. Embedding GPG is feasible if development is centralized [6], because it requires one release key. Distributed development however brings other challenges. In a distributed community developers have their own maintainer keys. Downloading packages requires not only to check the signature but also to check the origin of the key.

Conclusion The best way to ensure integrity is making use of GnuPGP which is a hybrid combination of both symmetric and public key cryptography and based on GPG. The majority of Unix distributions already uses this technology, and combining the most recent version of GnuPGP with public key management and package auditing provides the integrity users need. GPG covers digital signatures and transmission faults as an integrated solution.

2.4 Availability

2.4.1 Introduction

Making an update is one thing, making sure every system gets this update is another thing. Updates frequently fix security issues in programs. If a vulnerability is known, and the available patch not applied, the system is vulnerable for attackers. Therefore, updates should be easy to acquire. This also poses a threat. If an update mechanism can be influenced in not installing updates, the system will stay vulnerable. This threat originates from more than one direction. This can be done by attacking the update repository with a DoS attack, or preventing the client from accessing the repository. Attacking the repository itself is more effective as no system is able to perform an update, however this is hard to accomplish. The greater risk lies in a DNS spoofing attack on the client which is easier to perform and also makes the update server unreachable for this specific client.

2.4.2 Theoretical solutions

It is very difficult to defend against a DoS attack. If an update server is not reachable, it is difficult to see if the connection is intentionally blocked or the server is simply down. System administrators should always be aware of security updates and check their system if these updates are downloaded and installed. If not, they should contact the distributor if there is a problem. User interaction still remains the best solution for this problem.

Distributed mirrors Update mirrors should be spread globally among countries and not be dependent from a central location. This reduces the chance an update cannot be installed because the update server is unavailable.

Distributed transmission Repositories will endure a peak usage during a release of a system critical component like a service pack or kernel release. During this time the speed of updating decreases dramatically leaving the clients waiting on a critical update. The article *Large Scale Update Distribution with BitTorrent* describes a proposal to integrate the Bittorrent file transfer technology into the existing package distribution systems. This reduces bandwidth usage and improves security because packets are widespread between peers. Therefore it will be more difficult to insert malicious code.

Conclusion It is very difficult to defend against an availability attack. There is not a single solution which solves this issue, but all of the described methods contribute in defending against these risks.

2.5 Overall Conclusion

In this chapter we have presented several requirements for the issues of confidentiality, integrity and availability. From this we are able to reveal the demands for a workable and secure update system summarizing the following requirements:

- Auditing of packages by independent and trusted third party and public key management.
- TSL encryption to enforce confidentiality
- A public key system to authenticate sessions, signed packages for validity and integrity.
- It should not be possible to override the system and install unverified packages.
- Update system with a short release cycle.
- Repository mirrors distributed among countries, support for all parties and competitors.

3 Method of analysis

Introduction To analyse in which way the chosen applications and operating systems correspond to the abstract model, an method must be defined. This method can be used to attack the programs. The purpose of this method is to discover the flaws and security weaknesses in the update mechanisms. After the attack a conclusion can be made about the security of the mechanism and how this can be exploited.

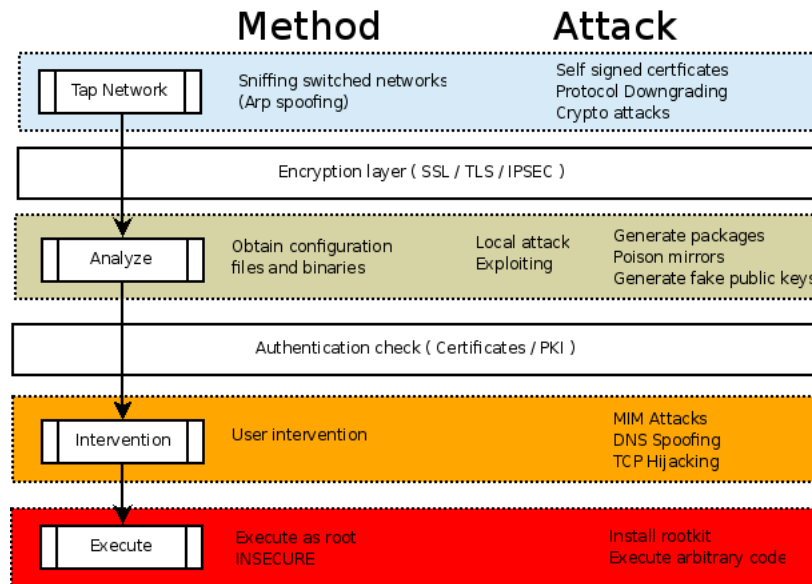


Figure 3: Paths of attack flowchart

Attack paths First we will describe the methodical ways in which an update mechanism can be attacked. The flowchart diagram above is slightly related to attack trees [22] to visualize these attack possibilities. The model describes that the attacker first need access to the (sub)network of the target system in able to eavesdrop the data connection. This access can be realized by cracking wireless access, for example a simple WEP connection or by tapping a physical link. Once the attacker is able to sniff, the attacker can also modify and manipulate packets in the data stream. If the connection is not encrypted using SSL, the attacker is able to obtain binaries and spoof sessions using DNS man-in-the-middle (MiM) attacks etcetera. If there is SSL encryption in place, this becomes a lot harder but not impossible. The success factor of a MIM attack depends on the use of signature checking using certificates (x509) or public keys (GPG). If there is no signature at all and no form of user intervention, the attacker in person is in; meaning that the target system can execute arbitrary code. Other attack vectors are also visualized, like poisoning update mirrors with trojanned packages or using any weakness (exploit) in the system itself.

According to the described model above, we execute the following path:

- Sniffing to discover the workings
- If there is an encryption layer, try to bypass it.
- Analyse packet information
- Try to discover an attack vector like DNS spoofing
- Upload and execute malicious code if possible

3.1 Summary of attacks

3.1.1 Confidentiality

Sniffing is a technique to intercept, log and analyse data streams on a network segment. Confidentiality is compromised when data is intercepted to obtain sensitive information, depending on encryption methods in use. Sniffing can be used for a variety of useful purposes, but in our case an attacker might derive how update systems work. Network protocols can be reverse engineered, passwords can be collected and communication can be debugged. To receive all traffic, the network interface is set to *promiscuous mode*.

Sniffing switched networks Normally when a system is connected to a switch, it only receives data which is meant for its network address. In order to capture all data from all systems on a network, the attacker can use a technique called ARP spoofing [27] [28]. Each switch has an ARP cache which contains information about who is connected to which port. By poisoning this cache, the attacker can claim himself as default gateway. With promiscuous mode enabled and ARP spoofing the attacker can capture all the data from the internal network.

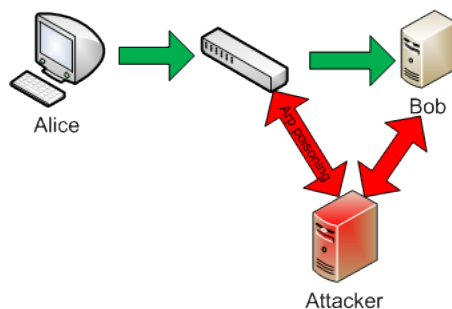


Figure 4: Arp spoofing

Social engineering is a serious security risk because it attacks the only component of a system that cannot be secured by firewalls and anti-virus software: the human brain. Social engineering is a collection of techniques to manipulate people to retrieve confidential information or computer access. Concerning update systems, it involves distributing trojanned media or convince users to install arbitrary software. Users should be aware of the software or updates that are installed on their system.

DNS spoofing DNS spoofing (described in the next section) can be used to eavesdrop on update mechanisms by leading the data transport through the attackers system.

3.1.2 Integrity

DNS Spoofing Most communication start by resolving IP addresses. This way an update server can change from IP address while still remain reachable for all its updateable applications. However, the DNS implementation is not without its flaws. As described in [23] it is very easy to let and DNS name point to another IP address. (DNS Spoofing)

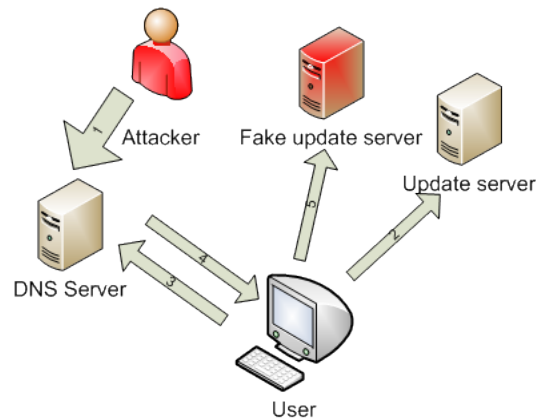


Figure 5: DNS spoofing

A way to test DNS spoofing is changing the hosts file on a local machine. By spoofing the DNS name, the attacker can redirect the update program to our own server and maybe even update the application with his own malicious code (Man-in-the-middle). The attacker stands between the system and update server. DNS spoofing can be a powerful attack method for hacking update mechanisms, if not all security demands are met.

TCP Hijacking TCP-connection hijacking is a form of MIM attack where a attacker can allow normal authentication between two hosts and seizes control over the connection. This can be done during a three way TCP handshake and in the middle of a established connection. It works by exploiting a desynchronized state in TCP communication so that both hosts will discard packets from each other. An attacker will then inject forged packets with correct sequence numbers with his own commands, known as a sequence number attack.

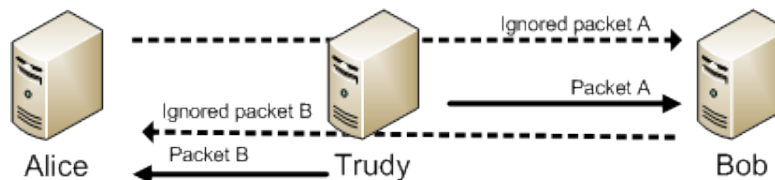


Figure 6: TCP Sequence number attacks

Uploading trojan packages Another scenario is to see if the attacker can affect existing update mirrors or upload own trojanned packages and patches. Especially open source development might be vulnerable for these types of attack because everybody can contribute their code.

3.1.3 Availability

One way of exploiting might be preventing new updates to be downloaded or installed. This document will refer to this as *denial of updates*. If the attacker can trick the update mechanism that there are no new updates available, none will be installed. This can be accomplished by performing a DNS spoofing attack for instance. By pointing the update server DNS name to another location, the application will not be able to download patches. This can then be exploited if there is a bug in the application, which is updated and fixed in an update package the attacker just prevented from installing. A system administrator might notice the prevention of an update if he pays attention to the released updates by a software distributor, but normally it will go unnoticed. Possibilities are to prevent the system from updating, or crash the update mechanism itself.

3.2 Consequences

When an update mechanism is cracked and its precise internal workings are known, vulnerabilities can be exploited. These are two methods to use this knowledge.

Local attacks Local attacks can be seen as an attack from the inside using a low privileged user account. It is interesting to see if a local user can get root privileges on a system using update system characteristics, vulnerabilities or exploits. Misconfiguration or some shell scripts/temporary files can permit a user to gain a higher level of access.

Local exploits are mostly found due to misconfiguration or wrong file permissions/ownership, buffer overflows or race conditions. Buffer overflows are mostly found in update systems written in C++/C and some other languages where boundaries (stack, heap) are not checked. Another way is to exploit race conditions, vulnerabilities which only exist in a limited time-frame.

Gaining access If the attacker has successfully cracked the update mechanism, it might be possible to update the system or application with his own program. This way the attacker can gain access to the system and able to get full control. This is the ultimate goal the attacker want to achieve when hacking an update mechanism and has the largest impact on a system.

4 Attacking in the field

4.1 Introduction and tools

The following utilities are used to launch several kinds of attacks on update mechanisms 3. The programs Wireshark and Paros are primary used for data gathering by sniffing data connections. Hashcalc is used to interpret various checksums and hashes used in proprietary update systems. Ettercap is the primary utility used for spoofing and denial of service attacks.

Wireshark is a widely know packet sniffer and protocol analyser for Unix and Windows platforms used to capture raw data from a network. It also contains a 'tcpdump' like console version named tshark. Early versions of Wireshark suffer from dozens of remotely exploitable security holes; so it's recommended to use the latest version. Wireshark has utilities to browse capture data, display filter language and view a reconstructed stream of a TCP session. It works by capturing packets through a library called libpcap. This library accesses the packets on the network through using packet socket mechanisms in the kernel.

Paros is a man-in-the-middle proxy server written in Java to monitor, log, capture, modify and resend HTTP(S) requests from client/server architectures. Paros listens on port HTTP port 8080, HTTPS port 8443 and uses it's own certificate to decrypt SSL communication. Paros is mostly used for vulnerability scanning, cookie manipulation or insufficient data validation by poisoning GET and POST variables. We use it to analyze HTTP(S) datastreams and modify requests, to see how update mechanisms react to MiM attacks.

HashCalc is a free Win32 based tool to compute multiple hashes, checksums and message authentication codes (HMAC) from files. We use it to identify applied hashing algorithms (MD2, MD4, MD5, SHA1, SHA2,SHA256, SHA384, SHA512) and checksums like CRC on update mechanisms.

<http://www.slavasoft.com/hashcalc/index.htm>

Ettercap is one of the most versatile open source network sniffer, spoofer and logger for switched networks available. It has the ability filter and collect passwords and data from unencrypted protocols, filtering TCP/UDP payload, scanning ports, OS fingerprinting and launch all kinds of denial of service or MiM attacks. It also supports spoofing of HTTP(S) connections by using fake certificates. Ettercap comes with a plugin library by itself and has a both commandline as ncurses interface. Embedded are hooking plugins like H02_troll (ARP reply spoofing tool), dns_spoof to intercept and spoof DNS queries and bashee to kill update sessions between two hosts. <http://ettercap.sourceforge.net/>

Dsniff is a alternative widely known collection of command line tools similar to Ettercap used for auditing and penetration testing. The passive sniffing tools are filesnarf, mailsnarf, urlsnarf (http requests) and webspy. Arpspoof, dnsspoof and macof are used to intercept layer 2 traffic and perform MiM attacks on the same network. SSHmit and Webmitm can be used for MiM attacks on SSH and HTTPS sessions.

4.2 Operating systems

4.2.1 Windows XP

Introduction Windows XP has two ways to keep the operating system up to date. One is through a website called Windows Update (currently version 6) and the other is a system program: automatic updates.

Security The Windows Update website uses an Active X control. While Windows Update is now considered reasonable secure, it still uses Active X. Active X is a programming interface, which has suffered from many security issues in the past. It remains to be seen if active X will be secure enough in the future to allow for safe updating.

The update request is done via HTTP [1]. After this, all communication is done through TLS v1.0. As with Windows Vista, Windows XP SP2 uses BITS download service to transfer the downloads. BITS service description:

Transfers data between clients and servers in the background. If BITS is disabled, features such as Windows Update will not work correctly.

The automatic update mechanism is similar to the website version, however it can only update recommended and important updates. It does not utilize Active X and therefore should be safer. It also retrieves its updates through the BITS service and thus also utilizes TLS to retrieve its updates. BITS also ensures the integrity of the update binaries although it is unknown which algorithm is used.

According to earlier research [1] the Windows Update mechanism uses authenticated binaries. It offers code signing based on X.509 certificates. This mechanism is used by the Windows update site and the automatic update feature.

Analysis Because the data was sent encrypted, passive sniffing did not reveal much about the update procedure. Only the first part of the communication process was visible.

Data about the user/computer is also sent to the Microsoft server which is visible in some parts of the following capture.

```
POST /windowsupdate/v6/redirect.asp?OS=5.1&Processor=x86&Lang=en&CurrentSite=6Live&SP=2&control=
5.8.0.2469&MUOptIn=true HTTP/1.1
Accept: /*/*
Accept-Language: nl
Referer: http://update.microsoft.com/windowsupdate/v6/default.aspx?ln=en-us
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; InfoPath.2;
.NET CLR 2.0.50727; .NET CLR 3.0.04506.30)
Host: update.microsoft.com
Content-Length: 24
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: MC1=GUID=b507dd66da72554abda577e33c2e618a&HASH=66dd&LV=20068&V=3;
WT_FPC=id=145.97.218.233-3906765392.29804573:lv=1181510945796:ss=1181510945796;
A=I&I=AxUFAAAAAAMCQAADDPVVyFOIihL/E65o38FtA!!&CS=100azE301y19e0302113o03;
MUID=AABA071F63FC424BA931E420221C0743; MSPartner2=
LogUser=fb524268-ee25-4044-bd8f-a4152bc299b3&RegUser=;
ANON=A=019373C1A72AEB918EF7BEF1FFFFFFF&E=56c&W=d;
NAP=V=1.5&E=512&C=wz72J1P8gIioBVE2nL5teI0hcqT5THUEnFySh881HUkV01ij8TFfpA&W=d;
msresearch=1; WT_FPC=id=145.97.218.233-3906765392.29804573:lv=1181711764078:ss=1181711764078
```

Just before the connection switches to TLS, the following data is received:

```
v6/windowsupdate/selfupdate/wuident.cab?0706191101 HTTP/1.1
Accept: /*/*
User-Agent: Windows-Update-Agent
Host: update.microsoft.com
***
```

We discovered this is sent before the TLS connection as the "BITS" service is activated. The TLS v1.0 connection is clearly visible in the Wireshark log:

```

⊕ Frame 150 (124 bytes on wire, 124 bytes captured)
⊕ Ethernet II, Src: Intel_87:de:aa (00:90:27:87:de:aa), Dst: Cisco_54:8f:fc (00:05:5f:54:8f:fc)
⊕ Internet Protocol, Src: 145.97.218.245 (145.97.218.245), Dst: 65.55.184.253 (65.55.184.253)
⊕ Transmission Control Protocol, Src Port: 1961 (1961), Dst Port: https (443), Seq: 1, Ack: 1, Len: 70
  Source port: 1961 (1961)
  Destination port: https (443)
  Sequence number: 1 (relative sequence number)
  [Next sequence number: 71 (relative sequence number)]
  Acknowledgement number: 1 (relative ack number)
  Header length: 20 bytes
  ⊕ Flags: 0x18 (PSH, ACK)
  Window size: 65535
  ⊕ Checksum: 0x13ea [correct]
⊕ Secure Socket Layer
  ⊕ TLSv1 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 65
  ⊕ Handshake Protocol: Client Hello

```

Figure 7: Windows Update uses TLS v1.0

From analyzing the wire capture we conclude the update mechanism works as follows:

- Request Windows Update page (only for windows update website)
- Send information about computer
- Receive directions for TLS connection
- Invoke TLS
- Receive update list
- Request updates
- Receive updates
- Check certificates
- Apply update

According to our findings, man-in-the-middle attacks cannot be used to successfully weaken the update mechanism security. This is because of the TLS encrypted connection, but also because of the update package signing which was also discovered in [1]. Confidentiality is ensured because of these security measures.

Automatic update checks periodically if there are new updates available. However, if the connection to the update server was not established in time, no warning is given to the user. If the update server is blocked from the system, it is possible to prevent a system from installing updates. The warning itself will still be visible though if a user invokes a manual update via the Windows update site. A warning is showed describing the server could not be reached. Also, the update servers are not geographically spread. The amount of update servers is limited and therefore availability cannot be guaranteed.

Item	Findings
Integrity of developed updates	unkown
Confidentiality of traffic	yes, TLS v1.0
Integrity of source repository	authenticated source
Integrity of transmitted packages	yes, BITS
Availability of updates	centralized repositories, denial of update attack applicable
Attack successful	yes, partial

Table 1: Summary of results

4.2.2 Windows Vista

Introduction Windows Vista utilizes more or less the same update mechanism as Windows XP. The Windows Update website is replaced by an integrated update program within Windows Vista. Also, the automatic updates application is replaced by this single program and is also capable of scheduled automatic updates.

Security Because Windows Vista is also proprietary software, not much is known about the internal workings of the update mechanism. All information must be acquired by researching the protocol itself. Much information can be acquired from the knowledge we gathered on Windows XP as it uses the same Windows Update version.

Analysis From Wireshark we discovered Windows Vista uses like Windows XP an HTTP request to discover whether there are new updates available.

```
HEAD /v6/windowsupdate/redist/standalone/MUAuth.cab?0706131310 HTTP/1.1
Accept: */*
User-Agent: Windows-Update-Agent
Host: download.windowsupdate.com
Connection: Keep-Alive
```

The Wireshark capture showed Vista also makes use of Windows Update version 6 which is the same as Windows XP. The following communications are encrypted through TLS v1.0 and can therefore not be monitored by Wireshark. This complicates the research on this protocol. Microsoft uses for Vista also its BITS service to download the updates from the update server [24]. BITS ensures the integrity of the update binary.

Windows Vista has by default Microsofts certificate [1] installed and uses this to communicate secure with the windows update server. As with Windows XP, man-in-the-middle attacks like DNS spoofing and TCP hijacking will therefore not work. Confidentiality is ensured.

If the user only relies on scheduled automatic updates and the traffic to the Windows Update server is blocked, the mechanism can be prevented from installing updates by blocking the windows update server from the system. It does not present a warning to the user if an update attempt fails. Only if the user checks the update process he can discover if the update mechanism is being altered. As windows Vista uses the same update servers as windows XP, availability is also not ensured.

Item	Findings
Integrity of developed updates	unkown
Confidentiality of traffic	yes, TLS v1.0
Integrity of source repository	authenticated source
Integrity of transmitted packages	yes, BITS
Availability of updates	centralized repositories, denial of update attack applicable
Attack successful	yes, partial

Table 2: Summary of results

4.2.3 Redhat package manager

Introduction Redhat Enterprise Linux (RHEL) is one of the oldest and most widely deployed Linux distributions in the business market. The company Redhat, Inc committed seven years support after every Redhat release and offered certification programs, while the source code remained free. Distributing of RHEL is not permitted due to its trademark restrictions and several groups compiled their own forks. Scientific Linux is one of those Redhat based distributions used in academic environments. CentOS (Community ENTERprise Operating System) is the most popular one and offers full compatibility with RHEL.

Security All Redhat based distros use the Redhat Package Management format (RPM) to distribute packages and the Yellow dog Updater (YUM) to perform updates. YUM is developed at the Duke University [29], written in Python and can handle both source RPM (SRPM) as binary packages. It replaces systems like URPMI and Up2date and has support for multiple repositories. However, dependency handling is not as sophisticated and error-prone as a system like APT (Advanced Packaging Tool), because RPM was never meant to resolve dependency issues. YUM relies on GPG (GNU Pretty Good Privacy) for host authentication and GPG signed binaries. GPG public keys are retrieved from the mirrors themselves using URL given in repository files.

```

#/etc/yum.repos.d/CentOS-Base.repo
[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os
#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/
gpgcheck=1
gpgkey=http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-5

```

Table 3: Basic YUM repository file

Analysis Retrieval of packages from remote packages is based on standard HTTP 1.1 GET requests. Those are obviously not encrypted, as shown in the Wireshark log below. YUM uses GPG for integrity and authenticity, therefore MiM attacks will not work.

```

DNS Standard query A ftp-stud.fht-esslingen.de
DNS Standard query response CNAME rhlx01.hs-esslingen.de A 129.143.116.10
TCP 52649 > http [SYN] Seq=0 Len=0 MSS=1460 TSV=585604 TSER=0 WS=7
TCP http > 52649 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
TCP 52649 > http [ACK] Seq=1 Ack=1 Win=5840 Len=0
HTTP GET /pub/Mirrors/centos/5.0/updates/i386/RPMS/vim-X11-7.0.109-3.e15.3.i386.rpm HTTP/1.1

```

Table 4: Wireshark log YUM

YUM can use HTTP, FTP and NFS mirrors and supports failover so rpms can be safely delivered. It has control over different architectures like SPARC, i386 or AMD, and designed to facilitate distributive automatic updates. The YUM repositories are mirrored are maintained by using tool like rsync, which recursively copies all changes over a SSH encrypted channel. Keeping maintainer builds that way is like building a tree with several master repositories. The disadvantage is that one hacked server in this chain will crack every YUM server below the subtree, which can mean compromising hundreds of servers. One weak link in the chain might get found by distributing false GPG keys and backdoor packages. YUM and other packagers already have the capability to automatically retrieve public keys from the pgp.mit.edu or other key servers. Clearly, those algorithms don't check the keys for trustworthy signs. We can conclude that that there is always a point in each design where everything is broken, regardless which security method is used.

Item	Findings
Integrity of developed updates	partly, lot of custom development: Redhat only checks mainstream packages
Confidentiality of traffic	no, plain HTTP traffic
Integrity of source repository	yes, GPG: but easily overridden: automatic public key retrieval=INSECURE
Integrity of transmitted packages	yes, GPG signing, unsigned packages can be installed though
Availability of updates	distributed among countries
Attack successful	poisoning key-mirrors will discredit developers like us, so we didn't tried.

Table 5: Summary of results

4.2.4 Debian Advanced Packaging Tool

Introduction The Advanced Packaging tool (APT) is the standard package management system used by Debian Gnu Linux [23] and derivatives (Ubuntu and others). It can update both source (deb-src) as binary files. According to the Debian popularity contest the repository counts over 53000 different packages. Currently it is one of the most functional systems available because it has extensive dependency support and the ability to failback and pin down installed packages. Tools for generating Debian packages can be made more efficient. The developer must often tweak install scripts, makefiles and rules. Debian package repositories (stored in sources.list) are decentralized and easy to setup, which is a good thing. There is also a security team ready to audit mainstream Debian packages for security issues. The testing, unstable, edge, edgy, multiverse, universe and backporting branches are not reviewed by the security team.

Security In 2003, both the GNU FTP mirrors were compromised and the FSF (Free Software Foundation) changed their policy by signing MD5 checksums with GPG keys. Since version 0.6, APT supports signature checking of the release file for all archives. APT uses apt-key to manage a keyring of GPG (Gnu Privacy Guard) keys. The release file Releases.gz contains both SHA1, SHA256 and MD5 checksums of all files in the current archive and are signed by these keys. The default installation includes keys from the trusted Debian archives, but distributed development requires keys to be added manually to the trust database. The keys expire after one year and security of those keys are dependent of the Web of trust. The administrator should realize that overriding warnings when the GPG hashes do not match can compromise the system.

Analysis Wireshark showed us that the HTTP 1.1 get requests from APT version 0.6 and higher are not encrypted. The updates are indeed GPG signed and it is the responsibility of the administrator to validate the GPG public keys.

Protocol Info

```
DNS      Standard query A security.ubuntu.com
DNS      Standard query response CNAME ubuntu.bit.nl A 213.136.29.196
TCP      3944 > www [SYN] Seq=0 Len=0 MSS=1460 TSV=643255 TSER=0 WS=2
DNS      Standard query response A 82.211.81.138
TCP      1687 > www [SYN] Seq=0 Len=0 MSS=1460 TSV=643256 TSER=0 WS=2
TCP      www > 3944 [SYN, ACK] Seq=0 Ack=1 Win=185344 Len=0 MSS=1460 TSV=29487722 TSER=643255 WS=5
TCP      3944 > www [ACK] Seq=1 Ack=1 Win=5840 Len=0 TSV=643279 TSER=29487722
HTTP     GET /ubuntu/pool/main/l/language-pack-en/language-pack-en_6.10+20070601_all.deb HTTP/1.1
```

Table 6: Wireshark log APT

Item	Findings
Integrity of developed updates	yes: Debian security team
Confidentiality of traffic	no, plain HTTP traffic
Integrity of source repository	yes, GPG: but easily overridden
Integrity of transmitted packages	yes, GPG signing, unsigned packages can be installed though
Availability of updates	distributed among countries
Attack successful	no, we found no further flaws in APT.

Table 7: Summary of results

4.2.5 Gentoo Portage Tree

Introduction Gentoo is a complicated and fairly new source based operating system that updates and compiles each source package after retrieving it using a system called Portage. Portage is the package management system used by Gentoo Linux written in Python and consists of two parts: the emerge and ebuild system. The ebuild scripts are written in bash and contain the scripts to retrieve, upgrade, configure, compile and install packages. The Gentoo mirrors are distributed among countries and have more than 25000 ebuild source packages in their repository. Emerge manages dependencies and execute ebuild packages and maintains the Portage tree installed on the system. Portage is the first package management system that fully collaborates with version control systems and optimizes the system with CFLAG compilation settings. Software can be stripped down by setting USE flags to indicate which features they want to include when installing or upgrading a system. The only disadvantages of Portage is that it does not handle reverse dependencies and that compiling from source takes a lot of time. There are precompiled binaries available, but installing and compiling a full system can take hours or sometimes days.

Security Every Gentoo package has a Manifest file with the checksum and filesize of all files contained in it. Before 2003 several trojans [23] were released as proof of concept that it was easy for a blackhat to infect files in Portage. Any user who had access to an rsync mirror might compromise a large number of Gentoo users, because there was no practical way to verify if those files were changed (gentoo-dev mailing list bug 5902 and 6356).

Analysis Since version 2.0.51 Portage also supports verification of GPG signatures in Manifest files. Every cracker modification to files will change the signature. It is rather suprising that Gentoo is the first distribution which 'strictly' checks GPG signatures. The administrator can even change the FEATURES variable to fully trusted, meaning that the GPG keys got to be signed as well.

Item	Findings
Integrity of developed updates	Yes, Gentoo security team & Bugzilla
Confidentiality of traffic	no, plain HTTP traffic
Integrity of source repository	yes, dependant of FEATURES variable for strict checking
Integrity of transmitted packages	yes, strict signing
Availability of updates	distributed among countries
Attack succesfull	no, we found no further flaws in Portage

Table 8: Summary of results

4.3 Applications

4.3.1 Mozilla Firefox and third party plugins

Introduction Mozilla Firefox is a open source cross-platform browser started as fork of the Mozilla application suite and has a browser market share of 14 percent. The platform is written in C++ but the firefox interface is written in XUL (User Interface Language) and JavaScript. XUL based applications use XPI (Cross platform install) packages to install third party plugins that add functionality to the main program. Extension and application update parameters can be set in *about:config*.

Security All Mozilla based applications are natively programmed with security in mind. It supports a wide array of security protocols and include the use of a sandboxed model and same origin policy.³ Firefox uses SSL to check the Mozilla servers for updates and the update server will return a Manifest file (in XML) with the URL to the latest update. A update is also retrieved through SSL and the integrity of the package is verified by computing a SHA2 hash over the package. If the hash match matches to the one included in the Manifest file it will automatically start the upgrade. In addition to this appearing fail-safe security scheme Mozilla rewards people with 500 dollar with reporting a critical security bug.

Analysis Although the update system looks secure, the latest news [8] reported vulnerabilities in the corner of third party plugins. To prevent malicious programs all sites other than Mozilla Addons are blocked by default, so that malicious sites cannot install XPI extensions without user confirmation. This is indeed true if they use SSL, but well known add-ons use(d) insecure connections including Google, Yahoo, Facebook, Netcraft anti-phising, AOL and many others. Without SSL it is possible to sniff on add-on update requests and fake the update site reponse with malicious malware.

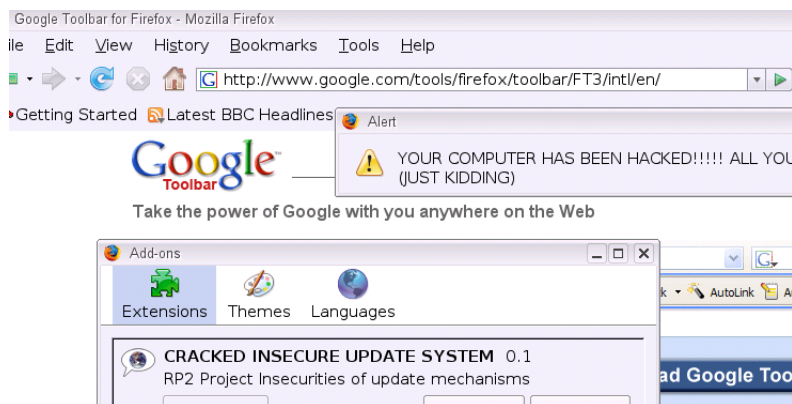


Figure 8: Spoofing attack on Google toolbar Firefox plugin.. [30]

With this knowledge we launched our own DNS spoofing attack 8 against Google Toolbar and verified this vulnerability. Users connected to WEP encrypted or unsecured wireless networks on public places were most vulnerable to this kind of attacks. This hack worked almost four weeks until the end of our research period, when Google moved their plugins to SSL secured servers. Our latest Wireshark log confirms that Google Toolbar now uses TLS encryption for their updates.

```
TCP      https > 1173 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1260
TCP      1173 > https [ACK] Seq=1 Ack=1 Win=65535 Len=0
TLSv1   Client Hello
TCP      https > 1173 [ACK] Seq=1 Ack=151 Win=8190 Len=0
TCP      [TCP Window Update] https > 1173 [ACK] Seq=1 Ack=151 Win=5720 Len=0
TLSv1   Server Hello, Certificate, Server Hello Done
TLSv1   Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
TLSv1   Change Cipher Spec, Encrypted Handshake Message
TLSv1   Application Data
TLSv1   Application Data
```

³security measure for Javascript that prevents setting properties from another origin

Item	Findings (Third party packages)
Integrity of developed updates	dependant of third party vendors
Confidentiality of traffic	dependant of third party vendors, normally SSL: yes
Integrity of source repository	no, not always: normally: yes
Integrity of transmitted packages	no, not always: normally: yes
Availability of updates	heavily centralized
Attack successful	yes, in the first week of this project.

Table 9: Summary of results

4.3.2 Java virtual machine

Introduction The Java runtime environment is a package containing the Java virtual machine that allows to execute Java applications. The Java virtual machine can be run on several operating systems and is completely processor and operating system independent. Sun provides an automatic update system to keep the virtual machine updated with the latest patches, This system called jusched.exe is automatically installed under Windows operating systems. If the update system finds a new update, it notifies the user and automatically install updates.

Security Java update uses X509 certificates with SHA1 signatures to verify whether the binary is signed by Sun or not. Many users will simply ignore this kind of warnings because they are used to all kinds of Windows warnings for installing third party drivers. In fact, Sun even recommends insecure updates for Java JRE 1.6 plugin of Firefox in their own documentation.

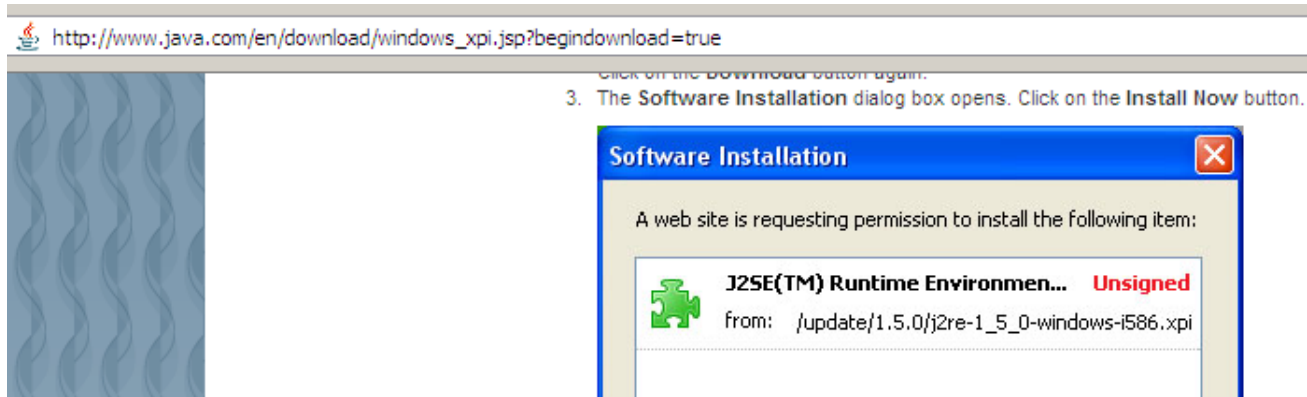


Figure 9: Sun recommends insecure updates.. [30]

One month ago (May-31-2007) the Google security team released patches for JRE to solve buffer overflow vulnerabilities in the imaging parsing code from JRE. The buffer overflow can be used for any version before JDK or JRE version 6 update 1 to allow a untrusted applet to gain privileges or cause denial of service. According to the description of this vulnerability [31] any applet hosted on any platform can read or write local files or execute applications from the current user. Sun released patches for solving these vulnerabilities. However, the update system will not uninstall previous versions of JRE and leaves the system vulnerable.

The Internet Storm center (SANS) released an article that the same vulnerability is already used in a existing Trojan. This trojan is a class file with a embedded GIF image used for a buffer overflow attack. It states that 86% of the users have Java installed and are vulnerable for this attack. The article also explains that most virusscanners will not detect this kind of malicious Java classes.

Analysis By sniffing the communication of JRE 1.5 we found that the session was unencrypted and not confidential. The original website of Sun Microsystems forwards sessions several times before sending the updates. All XML files from older versions point to the same new update.

```

Protocol Info
TCP 1273 > http [SYN] Seq=0 Len=0 MSS=1260
TCP http > 1273 [SYN, ACK] Seq=0 Ack=1 Win=34020 Len=0 MSS=1460
TCP 1273 > http [ACK] Seq=1 Ack=1 Win=65535 Len=0
HTTP GET /update/1.5.0/map-1.5.0.xml HTTP/1.1
TCP http > 1273 [ACK] Seq=1 Ack=153 Win=34020 Len=0
HTTP HTTP/1.1 302 Moved Temporarily
TCP 1273 > http [ACK] Seq=153 Ack=234 Win=65302 Len=0
TCP http > 1273 [FIN, ACK] Seq=234 Ack=153 Win=34020 Len=0
TCP 1273 > http [ACK] Seq=153 Ack=235 Win=65302 Len=0
TCP 1273 > http [RST, ACK] Seq=153 Ack=235 Win=0 Len=0

```

Table 10: Sniffing the DNS spoofing attack

We launched a DNS spoofing attack on a update from JRE 6 to JRE 6 Update 1, by creating a exact copy of the mirror and modified binaries. Once the user confirms, the binary executes and activates any arbitrary code. In our case, we programmed a simple platform independent Python application [10]

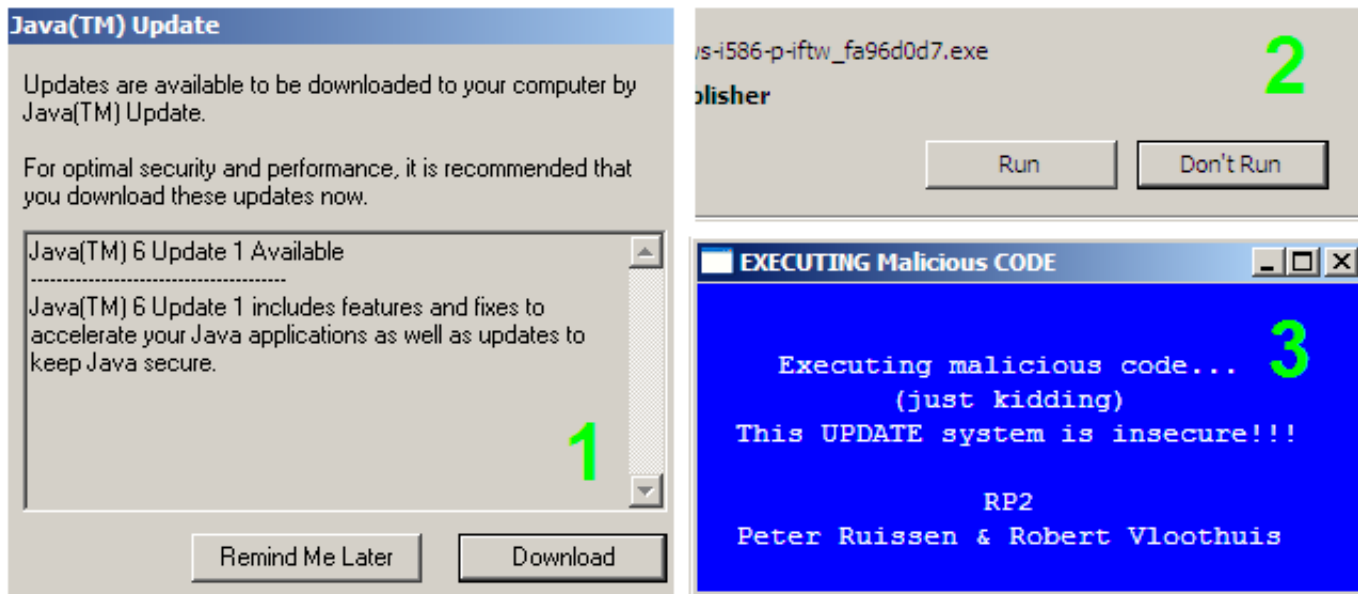


Figure 10: Java update spoofing attack

Most users run as administrator under Windows, so the program might silently install a virus while installing a modified version of JRE.

Item	Findings
Integrity of developed updates	unknown
Confidentiality of traffic	no, plain HTTP traffic
Integrity of source repository	no, unauthenticated source
Integrity of transmitted packages	yes, signing, unsigned packages can be installed though
Availability of updates	centralized repositories, denial of update attack applicable
Attack succesful	yes

Table 11: Summary of results

4.3.3 Adobe acrobat reader

Introduction Acrobat reader is a program by Adobe, which is able to read and print PDF files. It is widely used and installed on many (corporate) systems, as it is the standard for PDF reading. To keep this program up to date, Adobe has its own Adobe update program, which automatically checks whether there is a newer version available. It has its own downloader and can apply the patch without user interaction.

Security In [1] the update mechanism is researched and discovered it uses signed packages for secure updating. The Adobe key was found in the original install package which ensures the update packages originate from Adobe. It ensures only valid updates are applied. The precise algorithm and key length for this mechanism was not discovered because the updater is proprietary software. Its internal workings are kept hidden.

Analysis We researched the latest Adobe updater mechanism shipped with Adobe Acrobat reader 8.0. From the Wireshark logs we discovered an certificate revocation list was downloaded from Adobe (<http://crl.adobe.com/aum.crl>) and traces of a PKI infrastructure were found.

```
...].{.0...+.....7.....0...U.....0...+.....7.....
080503165941Z0..(.U.....0...0......ldap:///CN=Adobe%20AUM,CN=aumrootca,CN=CDP,
CN=Public%20Key%20Services,CN=Services,CN=Configuration,DC=AdobePKI,
DC=local?deltaRevocationList?base?objectClass=cRLDistributionPoint.
;http://aumrootca.adobePKI.local/CertEnroll/Adobe%20AUM+.crl..http://crl.adobe.com/aum.crl0...
+.....7.....0...0......ldap:///CN=Adobe%20AUM,CN=aumrootca,CN=CDP,
CN=Public%20Key%20Services,CN=Services,CN=Configuration,DC=AdobePKI,
DC=local?certificateRevocationList?base?objectClass=cRLDistributionPoint0
..*.H..
```

This confirms earlier research which claims Adobe uses certificates to protect its update binaries. This ensures the integrity of the files.

We also discovered a manifest file in XML format is downloaded from the Adobe site <http://swupmf.adobe.com/manifest/50/win/AdobeUpdater.upd>.

```
G H/ -QSJko96+DrVidGulTGAAD/8iq4FAruCOYRS6Rr4ciTOAaCmABsTsqbuntM/u/n+1ABUB8jsbf7w
Z3J8j5o0j1ui46+6jcmU5lwiPZ0yaDwjQ0c1QVJ6I3AJT8T+qreEHPsUI7jhzUJtmKVuguvMpj0
+Xwf0snZ2KHUeXSfsSQThQ= <?xml version="1.0" encoding="UTF-8" ?>
<Manifest auVersion="5.0">
  <Component category="plugin" name="reader8rdr-en_US">
    <DisplayName default="en_US">
      <en_US>Adobe Reader Updater plug-in</en_US>
    </DisplayName>
```

The first strings before the XML type declaration suggest some sort of signing. This cannot be confirmed as Adobes update mechanism is proprietary and closed source.

```
<File>
  <Url>http://ardownload.adobe.com/pub/adobe/acrobat/win/8.x/AUMProduct.aup</Url>
  <Size>311296</Size>
  <FileInfo>
    <Type>0</Type>
    <Executable>0</Executable>
    <Args />
    <ExecutableName></ExecutableName>
    <Destination>%AppDir%\AMT</Destination>
  </FileInfo>
  <Verification type="au40">
    <Data>E65RdfBuUaCo/FNIRXvTFB4GvpkU63sMS31fNF+gtkcMf2TtEMhDcWDoD05cua3u
Q4rq9sjsx1zzCcelnjiEv2yiWrNzV4dRx7XNTpYqYBkx+ExKezjtCFt0xyEUVMPUCjNAiL7yeHwtOA
y+H4pFwcdXpRgCBqg8YcTqG93J0q9Y= </Data>
  </Verification>
</File>
```

The data tag in the XML file contains a checksum of the downloadable package. This also prevents man-in-the-middle attacks because the code cannot be altered as the checksum would not match.

Also, the next file which was downloaded: `http://ardownload.adobe.com/pub/adobe/reader/win/8.x/8.0/misc/AdbeRdrSD80_all.msi` contains adobes certificate.

So according to our research, the Adobe update mechanism works in the following order:

- Get the signed manifest file which contains all updates
- Get certificate revocation list and apply it
- Download update with certificate
- Check certificate and apply update

During the whole update process all data was plain text visible. Confidentiality is therefore not a topic the Adobe update mechanism addresses.

The update list is signed and therefore it is not possible to prevent a system from updating. However, the scheduled update is a background process and is exploitable by blocking its access to the Adobe update server. The update application does not inform the user whether it is unable to contact the server. Only a manual invoked update reveals this.

Adobe has its own limited update servers on a single location. This will not ensure the availability of the updates. If the update network of Adobe is attacked, updating the application will not be possible.

Item	Findings
Integrity of developed updates	unkown
Confidentiality of traffic	no, plain HTTP traffic
Integrity of source repository	yes, authenticated source
Integrity of transmitted packages	yes
Availability of updates	centralized repositories, denial of update attack applicable
Attack successful	yes, partial

Table 12: Summary of results

4.3.4 Command & Conquer 3, Electronic arts

Introduction Games might not be the first thing to worry about when it comes to security. However, many home pcs, which are used for working from home, might have some games installed. These are probably installed by the employees son/daughter. Such a system might still contain important corporate data or has a VPN connection to the office. If a game introduces a security flaw, the data comes at risk. Command & Conquer 3 is such a game.

Security This game also utilizes its own update mechanism. New patches for game flaws are automatically downloaded and installed each time the user runs it. It is imperative for this update mechanism to also be secure. From our research we discovered this is not the case.

Analysis Passive sniffing was very easy as the connection to the update server was not encrypted. Confidentiality is not ensured. Sniffing the update procedure with Wireshark revealed the following steps to updating the game:

- Send version number to the (static DNS name) update server.
- Client receives a list in plain text with instructions where to download the new game files. This list contains the FTP server, file list and for each file the exact file size and hash.
- Client logs into the FTP server via the received DNS name. This FTP server allows anonymous access.
- For each file the size is checked through the FTP command SIZE before it is downloaded. Afterwards the hash is checked
- If one of those two requirements are not met, the update process fails.
- Client runs patchprepare.exe and patchupdate.exe which is also downloaded to finish the update process.

It seems there is not a single security measure taken to prevent updating the game with malicious code.

To demonstrate this vulnerability, we describe our own attack, which uses DNS spoofing. To make it simpler to analyze the process, we added servserv.generals.ea.com to the Windows hosts file and let it point to one of our own servers. This server only serves the file /servserv/cnc3/cnc3_english_1.4.patchinfo and contains our own modified text file with update instructions.

Wireshark capture of the HTTP request:

```
GET /servserv/cnc3/cnc3_english_1.4.patchinfo?value=1adb918 HTTP/1.1
User-Agent: ea_rts_launcher
Host: servserv.generals.ea.com
Cache-Control: no-cache
Cookie: com.pogo.unid=6456937868733571
```

```
HTTP/1.1 200 OK
Date: Fri, 08 Jun 2007 15:14:55 GMT
Server: Apache/1.3.26 (Unix)
Last-Modified: Thu, 07 Jun 2007 16:42:21 GMT
ETag: "3a273-1f0e-4668356d"
Accept-Ranges: bytes
Content-Length: 7950
Content-Type: text/plain
```

Fragments from the text file:

```
ftp://ftp.ea.com/pub/eapacific/cnc3/patch15
patch_self | | | |
run | patchprepare.exe | 99856 | 982eb047 |
```

On our own FTP server we hosted a complete mirror of the original EA update server. The update file is practically the same, only CNC3.exe is swapped with the file-size and hash from patchprepare.exe. The hash ensures the integrity of the file. Also on the FTP server we replaced CNC3.exe with patchprepare.exe. We could not swap CNC3.exe with our own malicious code, as we did not crack the hashing algorithm. As it is a relative easy algorithm with a very limited hash size, it will be easy for a serious attacker to crack. The update procedure resulted in having CNC3.exe replaced by the contents of patchprepare.exe, which is proof of the success of our attack.

If this .exe file where to be malicious and the game to be launched, the system might be compromised. The update is run on windows XP with privileges from the current user. Most of the time this is the administrator privilege.

It is also very easy to prevent a system from updating. As there is only a single FTP server which hosts the update binaries, it becomes the single point of failure. The availability of the update mechanism is not in order.

Item	Findings
Integrity of developed updates	unkown
Confidentiality of traffic	no, plain HTTP traffic
Integrity of source repository	no
Integrity of transmitted packages	yes, unknown CRC/hash algorithm
Availability of updates	single repository, denial of update attack applicable
Attack successful	yes

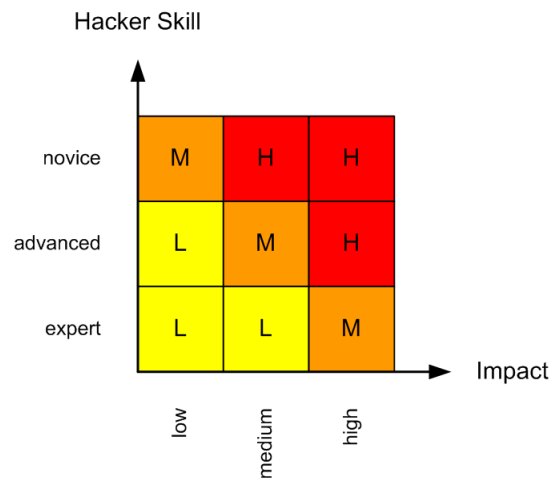
Table 13: Summary of results

5 Risk Assessment

Most updates are applied with elevated privileges because it needs file access to apply the patches. An update mechanism introduces a serious risk because it executes code with higher privilege than the user. If the mechanism is compromised and malicious code introduced it might have a high impact on the system.

The risk for a system to be compromised depends on the installed software, what kinds of vulnerabilities are known of this software and how hard it is to exploit these. If a vulnerability is discovered in a particular software packet, an evaluation must be done about what level an attacker should have to successfully exploit this fault. An easy to perform attack which has a high impact in combination with a low-skill attacker can be a high risk. However if the vulnerability is hard to carry out with a low impact, and the attacker has a low-skill, the risk would also be low.

To evaluate and categorize a risk, the following table can be used. This table displays the level of an attacker versus how heavy a system can be compromised. These factors combined can be labeled as a high, medium or low risk. The higher the impact and lower skill attacker required, the higher the risk.



The figure is a 3x3 matrix titled 'Risk assessment'. The vertical axis is labeled 'Hacker Skill' and has three levels: 'novice', 'advanced', and 'expert'. The horizontal axis is labeled 'Impact' and has three levels: 'low', 'medium', and 'high'. The cells in the matrix are colored and contain risk levels: 'M' (Medium), 'H' (High), and 'L' (Low). The colors are: orange for 'M', red for 'H', and yellow for 'L'. The risk level increases as the hacker skill decreases and the impact increases.

Hacker Skill	low	medium	high
novice	M	H	H
advanced	L	M	H
expert	L	L	M

Figure 11: Risk assessment

If we compare the results of our research with the risk assessment table, we come to the following conclusions:

Risk	Applies to	Knowledge x impact	Counter measures	Remaining risk
1. Denial of Updates	Windows XP/Vista	Advanced x Medium = Medium	Manual Update procedure	If user is only relying on the automatic update procedure, patches will not be applied
2. Hacking 1 GPG mirror, compromises all mirrors	Redhat	Expert x High = Medium	Revocation of GPG keys. Manually checking of keys.	GPG keys are not often updated. GPG keys are also automatically downloaded
3. Installing malicious updates from compromised mirrors	Debian	Expert x High = Medium	GPG signing	GPG warnings can be easily skipped
4. Updating of firefox extensions can lead to inserting malicious code. MIM attack required. No SSL is used	Firefox	Novice x High = High	SSL, but not used in many extensions	With a MIM attack, an extension can be updated with malicious code
5. Updates from java can be altered with a MIM attack	Java VM	Novice x Medium = High	Singing with certificates	Wrong certificate can be easily ignored. Older versions not deinstalled
6. Java plugin for firefox is unsigned	Java VM/Firefox	Novice x Medium = High	Check manual on certificates	Certificate is not signed and must manually be accepted
7. Adobe acrobat Denial of Updates	Acrobat Reader	Advanced x Low = Low	Manual update procedure	If a system only relies on automatic updates, updating of the reader can be blocked
8. Updating can be intervened with man-in-the-middle attack to install malicious code	Command & conquer 3	Advanced x Medium = Medium	Unknown hashing algorithm.	If hashing algorithm is cracked, update mechanism can be exploited

This evaluation can help system administrators determine the risk to their own system.

6 Conclusions

A lot has happened since the first article on update mechanism security. Many operating systems and applications have improved their update mechanisms with security measures like signed binaries and methods to securely transfer updates to the target system. However, we have proven there are still some applications like Java runtime environment and Command & Conquer 3 that have flaws in their mechanism which could be exploited. As most computer systems have several applications installed, one faulty update mechanism can be enough to compromise the entire system, and the risk of this happening is considered to be high. Therefore, each application should be investigated as to how the mechanism works and whether it is safe enough to use in its environment. On basis of these results a risk assessment can be made to investigate if the system can be compromised.

It is important for developers to give extra thought to designing the update mechanism. The abstract model established in this document can be used as a guideline for this task.

6.1 Recommendations

In this research various flaws in the security of update mechanisms were identified. To improve on this situation we advise to implement the characteristics of our established abstract model into an update mechanism.

By applying this abstract model to the security issues, we have been able to distinguish several recommendations. These can be made to two different groups: Developers and Users. Both groups can take action to improve security of update mechanisms.

6.1.1 Developers

In protecting the update mechanisms, developers should use the abstract model described in this report when designing update mechanisms. By using concepts like signed binaries with certificate (GPG), they can enforce integrity. Using SSL/TSL encryption can help them ensure confidentiality. Also, it might be a good idea to make a standardized open-source update system which can be used for every application or operating system. This way, more developers can have input in the design for this system, making it more secure. These measures will not ensure hundred percent security, however they will help make a system that is only hackable by highly skilled attackers.

6.1.2 Users

Where developers can make the update mechanism itself more secure, users can be more cautious when using such a system. For instance we have seen Java recommends installing an unsigned Firefox plug-in. Users should always check certificates and do not continue the installation if the source cannot be validated. Users should be aware of the risk when installing unverified packages. Also, update mechanisms which were proven to be unsafe, should be used with extreme caution.

6.2 Future work

Due to the short time provided for this research, a full and complete scan of the security risks for all chosen applications and operating systems was not possible. For instance, SSL man-in-the-middle attacks take more effort and time to analyze. However the fact that we found important flaws in Java, Command & Conquer and Firefox indicates there might be even more flaws in these and other applications should more time be available. Also, we were not able to research local attacks. It is however important to research these, since on a local system, an attacker might have more influence on an update mechanism even if this attacker has only user mode rights. By exploiting this mechanism it is likely possible to acquire elevated privileges by tricking it into downloading and executing malicious code under the elevated privileges of the update mechanism.

References

- [1] *Secure Software Updates: Dissapointments and New Challenges* Anthony Bellissimo, John Burgess, Kevin FU <http://prisms.cs.umass.edu/~kevinfu/papers/secureupdates-hotsec06.pdf>
- [2] *Secure Software Updates for Mobile Devices* Daniel Mettler, Ingenbohl <http://www.ifi.unizh.ch/egov/Mettler-TP.pdf>
- [3] *Large Scale Update Distribution with BitTorrent* Jarret D. Raim, Joseph S. Parker, Brian D. Davison <http://www.raimcomputing.com/temp/freenix-paper.pdf>
- [4] *Authentication Failure, Attacks in Software Update Mechanisms* Anthony Bellissimo, John Burgess <http://www.cs.umass.edu/~kevinfu/secureupdates/secureupdates-original-termppaper.pdf>
- [5] *Automatic update risks: can patching let a hacker in?* Kevin Dunn <http://www.sciencedirect.com>
- [6] *Security Issues of Auto-upgrades: apt-get install trojan* Jeff Covey <http://themes.freshmeat.net/articles/view/160/>
- [7] *A serious bug in Corel Linux update program allows gaining of local root* Tascon <http://www.securiteam.com/unixfocus/5VP0H000BY.html>
- [8] *A New Vector For Hackers: Firefox Add-Ons* Brian Krebs http://blog.washingtonpost.com/securityfix/2007/05/bungled_addon_updates_endanger.html?nav=rss_blog
- [9] *Vista PatchGuard Hacked* Marius Oiaga <http://news.softpedia.com/news/Vista-PatchGuard-Hacked-37979.shtml>
- [10] *Malware Update with Windows Update* Elia Florio http://www.symantec.com/enterprise/security_response/weblog/2007/05/malware_update_with_windows_up.html
- [11] *McAfee Aims for Microsoft's Jugular* Marius Oiaga <http://news.softpedia.com/news/McAfee-Aims-for-Microsoft-039-s-Jugular-36942.shtml>
- [12] *Yahoo's IM update: A Trojan horse of surprises* Stefanie Olsen, <http://news.softpedia.com/news/McAfee-Aims-for-Microsoft-039-s-Jugular-36942.shtml>
- [13] *Package Management System for Gentoo Linux* Simon Maynard <http://www.cs.bath.ac.uk/~amb/CM30076/projects.bho/2004-5/SimonMaynard-2004-5.pdf>
- [14] *HOWTO Infect Gentoo Linux Systems with a Trojan Horse* Alexander Holler http://linuxreviews.org/gentoo/gentoo_trojan_howto/
- [15] *Microsoft kills Net address to foil worm*, Ina Fried http://news.com.com/2100-1002_3-5064433.html
- [16] *XP SP2 Compatibility Testing Kit* Elizabeth Millard <http://www.processor.com/editorial/article.asp?article=articles%2Fp2637%2F41p37%2F41p37.asp>
- [17] *Providing Dynamic Update in an Operating System* Andrew Baumann, Gernot Heiser <http://ozlabs.org/~jk/projects/k42/dynamic-update-usenix.pdf>
- [18] *Limiting Vulnerability Exposure through effective Patch Management* Dominic Stjohn Dolin White <http://singe.za.net/masters/thesis/Dominic%20White%20-%20MSc%20-%20Patch%20Management.pdf>
- [19] *RPM* <http://wiki.rpm.org/>
- [20] *Providing Dynamic Update in an Operating System* Andrew Baumann, Gernot Heiser <http://ozlabs.org/~jk/projects/k42/dynamic-update-usenix.pdf>
- [21] *Limiting Vulnerability Exposure through effective Patch Management* Dominic Stjohn Dolin White <http://singe.za.net/masters/thesis/Dominic%20White%20-%20MSc%20-%20Patch%20Management.pdf>
- [22] *Attack Trees Modeling security threats* Bruce Schneier <http://www.schneier.com/paper-attacktrees-ddj-ft.html>
- [23] *DNS spoofing* Doug Sax http://www.giac.org/practical/gsec/Doug_Sax_GSEC.pdf

- [24] *Vista BITS* Microsoft <http://msdn2.microsoft.com/en-us/library/aa362781.aspx>
- [25] *[Yum] Security issues in yum in general...* Robert G. Brown <https://lists.dulug.duke.edu/pipermail/yum/2003-October/002299.html>
- [26] *RPM* Ladislav Bodnar <http://distrowatch.com/dwres.php?resource=article-rpm>
- [27] *An introduction to arp spoofing* Sean Whalen http://www.rootsecure.net/content/downloads/pdf_downloads/arp_spoofing_slides.pdf
- [28] *Sniffing in a switched network* Manu Garg http://www.infosecwriters.com/text_resources/pdf/arp_spoofing.pdf
- [29] *Duke University. 2002. Yellow dog updater* duke.edu/projects/yum
- [30] *Sun recommends insecure updates in own documentation* http://www.java.com/en/download/windows_xpi.jsp?begindownload=true
- [31] *Java vulnerabilities* <http://www.sunsolve.sun.com/search/printfriendly.do?assetkey=1-26-102934-1>
- [32] *Network Security* Charlie Kaufman, Radia Perlman, Mike Speciner <http://books.google.nl/books?id=KGkKAAAACAAJ&dq=network+security+private+communication+in+a+public+world>

Appendix

```
#!/usr/bin/env python
#!/usr/bin/python
# fonts.py

import wx

class MyFrame(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, wx.DefaultPosition,
wx.Size(320, 288))

        panel = wx.Panel(self, -1)
        text1 = "Executing malicious code...\n(just kidding)\n
This UPDATE system is insecure!!!\n
\n RP2 \nPeter Ruissen && Robert Vloothuis"
        font1 = wx.Font(10, wx.FONTFAMILY_MODERN, wx.NORMAL, wx.NORMAL)
        panel.SetBackgroundColour('BLUE')
        panel.SetForegroundColour('WHITE')
        lyrics1 = wx.StaticText(panel, -1, text1,(10,30), style=wx.ALIGN_CENTRE)
        lyrics1.SetFont(font1)
        self.Center()

class MyApp(wx.App):
    def OnInit(self):
        frame = MyFrame(None, -1, 'EXECUTING Malicious CODE')
        frame.Show(True)
        self.SetTopWindow(frame)
        return True

app = MyApp(0)
app.MainLoop()
```

Table 14: Python test application