Absolute Pwnage: A Short Paper About The Security Risks of Remote Administration Tools

Jay Novak, Jonathan Stribley, Kenneth Meagher, and J. Alex Halderman

The University of Michigan
{novakjs, stribs, meagherk, jhalderm}@umich.edu

Abstract. Many IT departments use remote administration products to configure, monitor, and maintain the systems they manage. These tools can be beneficial in the right hands, but they can also be devastating if attackers exploit them to seize control of machines. As a case study, we analyze the security of a remote administration product called Absolute Manage. We find that the system's communication protocol suffers from serious design flaws and fails to provide adequate integrity, confidentiality, or authentication. Attackers can exploit these vulnerabilities to issue unauthorized commands on client systems and execute arbitrary code with administrator privileges. These blatant vulnerabilities suggest that remote administration tools require increased scrutiny from the security community. We recommend that developers adopt defensive designs that limit the damage attackers can cause if they gain control.

1 Introduction

Remote administration products allow system administrators to manage collections of machines from a central location. These tools carry inherent security risks. If an attacker can exploit them to issue unauthorized commands, he may be able to take control of client machines. The question is, do the designers of remote administration software take adequate steps to protect the security of their users?

As a case study, we analyzed the security of Absolute Manage [1], a remote administration tool by Absolute Software. Absolute Manage has has been deployed by companies, universities, and school districts throughout the country [1]. It has been in the news since February 2010, when a school district in Pennsylvania was alleged to be using it to spy on students at home via their laptop webcams [17]. We selected it for our study after this controversy brought it to our attention. We had no reason to believe its security would be especially weak or strong.

We began with black-box testing, through which we determined that an attacker with control of the network could subvert local clients and run arbitrary code. Following this initial result, we used the IDA Pro disassembler [8] to understand the software's communication protocol and security mechanisms, which we found to contain significant design flaws. These vulnerabilities allowed us to develop attacks that require less control over the network while still providing complete adversarial control over the client.

The problems we found in Absolute Manage suggest lessons for remote administration products more broadly. We observe that the design flaws we uncovered were elementary mistakes that would not have gone unnoticed in even the most basic security review. Given the magnitude of the risks these products pose when they are vulnerable, we recommend that developers adopt defensive design and programming practices. The goal should be to ensure that, even if an attacker is able to issue unauthorized commands through the software, the damage he can cause will be limited.

Outline Sections 2 and **??** describe Absolute Manage and the software's communications protocol. Section 3 describes serious vulnerabilities we found in its encryption and authentication. Section 4 explains ways attackers could exploit these flaws to take control of clients. Section 5 discusses ways to mitigate the problems and draws broader security lessons. We survey related work in Section 6, and we conclude in Section 7.

2 Background

Absolute Manage is a product of Absolute Software, which purchased it from Pole Position Software in December 2009 for $12.1 million and 500,000 shares of common stock [2]. (Prior to the acquisition, the software was called LANrev.) Using server-side tools, administrators can instruct clients to install programs, apply software updates, run scripts, take screenshots, or execute code, among other functions. Clients also report status information to the server at regular intervals (by default, every 15 minutes). A feature called TheftTrack can be activated to cause the client to send a screenshot and an image taken with the computer's camera along with each status report. The client software supports Windows and Mac OS X.

One place where Absolute Manage is used is Lower Merion School District in eastern Pennsylvania, which installed it on laptops issued to around 1800 high school students. In February 2010, one of those students, Blake J. Robbins, sued the district in federal court, alleging that school officials violated students' privacy rights by secretly using the laptop cameras to photograph them in their homes [17], using the Absolute Manage TheftTrack feature.

The Absolute Manage software suite includes Absolute Manage Agent, which runs on client machines, and Absolute Manage Server. Both sides accept TCP connections; by default, clients listen on port 3970 and servers listen on port 3971. Clients automatically contact the server at a configurable "heartbeat" interval to report information about system state. The server can issue commands as a response to the heartbeat or by directly contacting the client. If a command cannot be delivered, it is queued and resent in response to the next heartbeat.

The body of each message is an XML-formatted property list. The properties include:

– *AgentSerial* A unique identifier for the client that is randomly generated on installation.

- *AdminUUID* A unique identifier for the server that is randomly generated on installation.
- *CommandUUID* A unique identifier for the command that is randomly generated when it is issued.
- *CommandID* A number that specifies the command to be executed and determines the format of the *CommandParameters* structure.
- *SeedValue* A 192-bit binary value used to authenticate the server.

Prior to transmission, all messages are compressed with `zlib` and encrypted using the Blowfish cipher [19] operating in ECB mode.

3 Vulnerabilities

Due to a number of design flaws, the Absolute Manage protocol fails to provide adequate integrity, confidentiality, or authentication.

3.1 Defective Encryption

The Absolute Manage developers opted for a simple cryptographic design: all clients and servers use the same hard-coded secret keys every time, for every message. We were able to discover the keys by examining the client program with IDA Pro. There are at least four keys used for different purposes: two are based on a German phrase and differ only in punctuation, one is a minor corruption of a common colloquial expression, and one appears to be a snide remark about a design choice. All can be exposed by running the `strings` command on the client binary.

Using hard-coded secret keys is highly risky, since an attacker who manages to extract them from one copy of the software can then attack all the other copies. In the case of Absolute Manage, an attacker who learned the keys could decrypt intercepted protocol messages, including messages containing sensitive private data or proprietary software; he could act as a man-in-the-middle and arbitrarily modify the contents of messages in transit; or he could generate new encrypted messages from scratch and pass them to servers and clients. Another important flaw is that the protocol uses ECB mode, so blocks are encrypted independently and deterministically. This lets attackers compromise the protocol even without knowing the keys.

3.2 Defective Authentication

Since remote administration software gives parties the ability to control the machine, it is essential to ensure that only *authorized* third parties can do so. Unfortunately, Absolute Manage uses an extremely weak authentication mechanism for its client-server communication.

When the client receives a command message, it tries to confirm that it originated from an authorized server. Each server has a unique SeedValue that

it includes in all its command messages, and clients discard commands that do not have the expected value. Clients will accept any properly-formatted message aslong as it includes the correct SeedValue. The AdminUUID, AgentSerial, and CommandUUID properties can have arbitrary values.

Since the SeedValue property is a random-looking 192-bit number, one might expect it to be difficult to guess, but, in fact, it carries very little entropy. If we decrypt the SeedValue using Blowfish in ECB mode and a different hard-coded key, we get the following bytes:

```
00 00 00 0E 00 31 00 34 00 30 00 31
00 34 00 37 00 35 00 00 00 00 00 00
```

The first four bytes are a length specifier, and the trailing zeroes are padding. After removing these, we are left with the UTF-16 encoding of a 7 character string: `1401475`. This is the server's "serial number," which was provided by Absolute Software along with the product activation key when we purchased our license. If all server serial numbers are 7 digits like ours, and they are randomly assigned, then the SeedValue property contains about 23 bits of entropy. We suspect the assignment is nonrandom, so the actual entropy may be much less.

In certain situations, clients do not apply any authentication at all. One case is when the client originates the connection. The server normally does not send SeedValue in its response, and the client does not check for one.

Another case is upon client initialization. The client discovers its server's SeedValue by *asking the server*. It sends a heartbeat message with the NeedSeedValue property set to true. Until the client receives a NeedSeedValue response from the server, it defaults to accepting commands with any SeedValue. Clients do not store the SeedValue to disk, so they need to ask the server again every time the software starts.

An additional vulnerability is that the servers do not authenticate messages from clients. Any correctly formatted message sent to the server is processed as a valid message, no matter where it originated or who sent it. Any client can send the server a heartbeat message with NeedSeedValue set to true, and the server will respond with its SeedValue.

The encryption and authentication mechanisms in Absolute Manage are deeply flawed. In the next section, we will describe ways that adversaries might exploit these vulnerabilities to carry out attacks against users.

4 Attacks

In this section we discuss how attackers can exploit the vulnerabilities we described to take control of client systems. We divide these attacks in two classes: on-path attacks, which require the attacker to be able to see the packets from the victim or the server, and off-path attacks, which can originate from any Internet location. An attacker might use any of these methods to gain the ability to issue unauthorized commands to an Absolute Manage client. Most generally, it allows him to silently install and run arbitrary code with administrative permissions.

4.1 On-Path Attacks

An attacker who can observe Absolute Manage traffic can use a number of techniques to identify clients to target and to learn the server's SeedValue, with which he can issue arbitrary commands to all the server's clients. The most basic attack is to listen for connections initiated by the server, which can be recognized by the default client port number. The destination IP address identifies a potential victim, and, since server-originated commands always contain the SeedValue, the attacker can decrypt them to learn it. The attacker can also send his own client heartbeat message to the server with the NeedSeedValue property set to true. The server will unwittingly send its SeedValue back to the attacker. Similarly, an attacker can issue commands to the client without knowing the SeedValue by mounting a TCP hijacking attack [18].The attacker can listen for a heartbeat and then forge the server's reply by spoofing packets from the server's IP address. (He can learn the correct sequence number by observing earlier packets in the connection.) An attacker can exploit the software's poor authentication by redirecting the client's connection to an Absolute Manage server he controls, thus allowing the attacker to take control of the client without knowing either the hard-coded keys or the SeedValue. Finally, some clients are configured to contact servers using a hostname instead of an IP address, and attackers can redirect their connections by using DNS cache poisoning [5].

4.2 Off-Path Attacks

There are several other attacks against Absolute Manage clients that can be performed by adversaries anywhere on the Internet. These attacks can target any client with a publicly accessible IP address. If the attacker knows the IP address of a client running Absolute Manage, he can use a brute-force attack by sending commands to the client with different SeedValues until one of them turns out to be correct. Since the SeedValue is a function of the server's serial number, the search space is relatively small. Each server uses the same serial number for all its clients, so after the attacker guesses it for one client, he can compromise all the server's other clients without any additional guesswork.

An attacker who wants to cast a wide net can use a different attack to efficiently target all clients at once. The first stage of the attack is to build a dictionary of server SeedValues. Servers running Absolute Manage have a unique signature that can be identified using a port scanner such as `nmap` [6]. With such a tool, an attacker can perform an Internet-wide scan for publicly addressable servers. Whenever he finds a server, the attacker sends it a heartbeat message asking it to return its SeedValue, and he adds the response to his dictionary. In the second stage of the attack, the attacker performs further network scanning to locate clients and uses the SeedValue dictionary to mount rapid guessing attacks against them. Using this attack, an adversary can take control of a large fraction of the publicly addressable machines running the Absolute Manage client.

5 Defenses and Lessons

Absolute Manage users need to take immediate steps to protect themselves from the attacks we have described. For the rest of us, the problems in Absolute Manage carry lessons about security risks in remote administration software more generally and about patterns of security failure.

5.1 Risks of Remote Administration Tools

Remote administration products like Absolute Manage carry large risks because they intentionally create mechanisms that allow remote parties to take control of a machine. There will always be a risk of *abuse* by authorized parties, as alleged in the students' lawsuit against Lower Merion School District, but correctly designed technology should at least prevent unauthorized third-party attacks by making sure only authorized parties can issue commands. This requires getting authentication right—exactly what Absolute Manage failed to do.

Because of these inherent dangers, remote administration software warrants careful security scrutiny during design, implementation, and testing. Furthermore, remote administration software should be designed defensively in order to minimize the harm to users if the authentication does fail. For example, clients could default to allowing only a minimal set of low-risk operations, and enabling additional operations could require physical access by an administrator. Or, if the client was intended to allow software installation but not remote desktop control, it could be designed to only allow the installation of binaries signed with a secondary key controlled by the administrators.

When remote actions do take place, clients should give users prominent notification and even a chance to cancel or postpone the activity. Howell and Schechter [7] recently proposed a UI paradigm for giving users control over sensor data that might be applicable in this context. Keeping users informed about what is happening to their computers would make attacks—and abuse—easier to detect and avoid.

5.2 Hard-Coded Keys as a Vulnerability Pattern

The problems with Absolute Manage are part of pattern of security failures involving hard-coded cryptographic keys. Using hard-coded secrets often negates the benefits of cryptography. It is widely recognized as a recurring vulnerability pattern, and was named one of the CWE/SANS "Top 25 Most Dangerous Programming Errors" for 2010 [4].

Why do developers keep making this mistake? One explanation is that using hard-coded keys allows developers to be able to say they use encryption while avoiding the complexity and expense of key management infrastructure. This problem is exacerbated by software customers who are unable to test the security themselves (and who may not want to go through the hassle of key management either). Customers usually will not realize that they are vulnerable unless they are attacked and they detect the intrusion.

Yet the cost of managing keys does not explain the whole problem. Absolute Manage did not simply forgo using PKI, it suffered from severe design flaws in almost every aspect of its use of cryptography. A strikingly similar example is the infamous Diebold AccuVote TS voting machine [11], where every vote record in every machine was encrypted using this DES key:

```
#define DESKEY ((des_key*)"F2654hD4")
```

Both systems employed deeply flawed, amateurish cryptography that provided almost no actual security. (Notice that in both cases the developers chose to use a string of text for the key!) Both were acquired from smaller companies with the vulnerabilities already present, yet neither purchaser managed to correct the flaws before they were discovered by others 6–18 months later.

Given that the broader security community treats the use of hard-coded keys with deserved contempt, we might conclude that it is a symptom of broader problems, indicative of companies that are devoid of security culture, process, or training. Yet perhaps this instead reflects a rational choice on the part of developers in light of the "weakest link" nature of security. Suppose you are a small developer with the resources to invest, at most, 50% of the cost of building strong security. Since investing 50% will likely leave the system just as vulnerable as if you had invested 1%, why waste the extra money? Retrofitting security is much harder than building it into a product from the start. If our hypothetical developer later sells the system to a larger company that can afford security in its home-grown products, the purchaser may nevertheless be unable to afford the investment needed to make the system secure.

Under this theory, hard-coded keys are a manifestation of a bimodal phenomenon that causes some rational developers to invest heavily in security and others to essentially give up on it. This suggests that the only way to prevent problems like those in the AccuVote and Absolute Manage is to change developers' incentives, either by making security much cheaper for them to build or by increasing the odds that insecurity will harm their profits.

6 Related Work

Past work has examined risks in Absolute Manage and in the broader class of remote administration tools.

On Absolute Manage To our knowledge, the first published analysis of Absolute Manage appeared in a blog post by security consultants Aaron Rhodes and stryde.hax [21] in February 2010. Their focus was the covert monitoring capabilities allegedly abused by administrators at Lower Merion High School, but they also expressed significant doubts about the program's security.

Other prior work came after we had finished our investigation but before we disclosed our results to the vendor. In late May, the Threat Level blog reported [24] that researchers from the Leviathan Security Group had discovered the hard-coded keys and demonstrated how they could be used to attack clients

on local networks. The researchers have not published the details of their findings, but we infer that their attacks cannot target remote victims.

Following the Threat Level post, we believed that real attackers would soon find the keys and discover the more powerful attacks we have described. To advise users of the danger, we posted an accessible summary of our results [15] on the Freedom-to-Tinker blog.

Other Administration Tools We know of one other instance where hard-coded keys in a remote administration tool led to security problems. In 2009, Symantec Altiris Notification Server was found to be using a hard-coded encryption key to protect a local database of sensitive credentials [22]. This exposed the credentials to discovery by unprivileged local users.

Another remote administration tool by Absolute Software reportedly contains security problems related to authentication. Computrace [1], a theft tracking and recovery tool, uses proprietary code in the system BIOS to resist removal. This support can apparently be exploited by an attacker to make malware harder to detect and remove [16].

Other tools that provide remote desktop control have adopted design philosophies that differ from the approach used in Absolute Manage. Apple Remote Desktop [3] and the Windows Remote Desktop Connection feature [13] normally display prominent notifications that the system is under remote control.

In contrast, Back Orifice [20] is a remote administration program that is designed to hide from users. Due to its potential for malicious use, a number of antimalware programs have added it to their blacklists. Indeed, the distinction between remote administration tools and malware like spyware, back doors, and bots can be a fine line—sometimes, the only difference is who is intended to be in control.

Another class of administration tools where security problems have been found is package managers, which are used to install and maintain software. Cappos et al. [9] examined ten popular package managers and found that all were had vulnerabilities related to package authentication and integrity protection. Some of these vulnerabilities could be exploited to run arbitrary code on client machines.

Lest we forget, the Internet's oldest tool for remote administration, `telnet` [10], used no cryptography at all. Incredibly, it took nearly 30 years for a secure alternative [23] to catch on. SSH has not been without its share of security problems [14]. Its inherent risks have inspired a range of secondary defensive mechanisms, such as port knocking [12].

7 Conclusions

In this paper, we revealed critical security vulnerabilities in Absolute Manage and demonstrated how attackers could harness them to cause widespread damage. We used these problems as a case study to discuss the broader risks of remote administration software and how they might be mitigated. The blatant vulnerabilities

in Absolute Manage suggest that this class of remote administration programs requires greater security scrutiny, particularly in light of the danger such software can pose when commanded by unauthorized third parties. Secure authentication is a necessity, of course, but such products should be further strengthened by employing defensive design and programming techniques.

References

1. Absolute Software. Absolute Manage Web Site. http://www.absolute.com/en_GB/products/absolute-manage.
2. Absolute Software. Absolute Software Acquires LANrev. http://www.absolute.com/company/pressroom/news/2009/12/lanrev, December 3, 2009.
3. Apple. Remote Desktop 3. http://www.apple.com/remotedesktop/.
4. CWE/SANS. 2010 Top 25 Most Dangerous Programming Errors. http://cwe.mitre.org/top25/.
5. Dagon, D., Provos, N., Lee, C.P., and Lee, W. Corrupted DNS resolution paths: The rise of a malicious resolution authority. *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS),* 2008.
6. Lyon, G. F. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.* Insecure, USA, 2009.
7. Jon Howell and Stuart Schechter. What You See is What They Get: Protecting Users from Unwanted Use of Microphones, Cameras, and Other Sensors. *Web 2.0 Security and Privacy,* 2010.
8. The IDA Pro Disassembler and Debugger. http://www.hex-rays.com/idapro/.
9. Cappos, J., Samuel, J., Baker, S., and Hartman, J.H. A Look in the Mirror: Attacks on Package Managers. *15th ACM conference on Computer and Communications Security (CCS),* pages 565–574, 2008.
10. Postel, J., Reynolds, J., and Reynolds, J. Telnet protocol specification. STD 8, RFC 854, May 1983.
11. Tadayoshi Kohno., Adam Stubblefield, Aviel Rubin, and Dan Wallach. Analysis of an Electronic Voting System. *IEEE Symposium on Security and Privacy,* pages 27–42, 2004.
12. Krzywinski, M. Port Knocking: Network Authentication Across Closed Ports. *SysAdmin Magazine,* 12(6):12–17, 2003.
13. Microsoft. Connect to Another Computer Using Remote Desktop Connection. http://windows.microsoft.com/en-us/windows-vista/Connect-to-another-computer-using-Remote-Desktop-Connection.
14. Provos, N. and Honeyman, P. ScanSSH: Scanning the Internet for SSH servers. *16th USENIX Systems Administration Conference (LISA),* 2001.
15. Jay Novak, Jon Stribley, and J. Alex Halderman. School's Laptop Spying Software Exploitable from Anywhere. Freedom-to-Tinker. http://www.freedom-to-tinker.com/blog/jhalderm/schools-laptop-spying-software-exploitable-anywhere, May 2010.
16. Alfredo Ortega and Anibal Sacco. Deactivate the Rootkit: Attacks on BIOS Anti-Theft Technologies. *Blackhat,* 2009.
17. Blake J. Robbins and others. Complaint Against Lower Merion School District Et Al. http://docs.justia.com/cases/federal/district-courts/pennsylvania/paedce/2:2010cv00665/347863/1/, February 16, 2010.

18. Bellovin, S.M. A Look Back At Security problems in the TCP/IP protocol suite. *20th Annual Computer Security Applications Conference,* pages 229–249, 2004.
19. Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). *Fast Software Encryption,* pages 191–204, 1993.
20. Sir Dystic. Back Orifice. http://www.cultdeadcow.com/tools/bo.html.
21. stryde.hax and Aaron Rhodes. The Spy At Harriton High. http://strydehax.blogspot.com/2010/02/spy-at-harrington-high.html, February 2010.
22. Symantec. Symantec Altiris Notification Server 6.X Static Encryption Key. http://www.symantec.com/business/security_response/securityupdates/detail.jsp?fid=security_advisory&pvid=security_advisory&year=2010&suid=20100128_00, January 2010.
23. Ylonen, T. SSH–secure login connections over the Internet. *Proceedings of the 6th USENIX Security Symposium,* pages 37–42, 1996.
24. Kim Zetter. School Spy Program Used on Students Contains Hacker-Friendly Security Hole. Threat Level. http://www.wired.com/threatlevel/2010/05/lanrev/, May 2010.