RoboAdmin: a different approach to remote system administration

Abstract. The most widespread approach to system administration consists in connecting to remote servers by means of a client-server protocol. This work analyzes the limitations of such approach, in terms of security and flexibility, and illustrates an alternative solution. The proposed model eliminates the predictable management port on the server by introducing an additional system, placed in between the remote server and its administrator, which allows to devise a management service behaving as a network client instead of a server. While the focus of this work is placed on the alternative communication model, whose effectiveness has been experimentally validated, the resulting architecture can be seen as part of a larger picture, tracing a research path leading to the definition of a novel administration framework, aimed at overcoming many other issues of the current techniques.

Key words: system administration, security, peer to peer networks.

1. Introduction

System administrators usually perform their tasks by connecting to a service allowing to control a remote system like it were local: examples of these services encompass the SSH daemon for Unix-like systems and various remote desktop solutions. This approach is so "natural" for the average administrator that he/she usually does not see its significant security and efficiency limitations. With this paper, we start a research line hopefully leading to overcome two issues: first, the lack of flexibility provided by the methods commonly available to the administrator for contacting the remote server, and second the lack of expressiveness usually associated to the command and control environment present on the server.

In this work, the focus is placed on the communication-related aspects, proposing an alternative to the common client/server model. A service implementing the latter kind of model has to listen to a known network port and to authenticate the incoming connection attempts. In many cases, the service has to be reachable from unpredictable locations, with standard client software; for this reason, implementing protection mechanisms relying on network tools (such as packet filtering) or on peculiar client capabilities (such as cryptography-based authentication) is not always viable. Both brute force attacks against the remote administration service and attempts at exploiting software/protocol vulnerabilities, then, must be considered possible. Host- and network-based intrusion detection systems can help thwarting the attacks before they succeed, but can not guarantee security against the vastly distributed attacks that are presently possible, especially considering the value of the target (that is full control of an Internet host).

The goal of this research is to devise an unconventional model of communication between the system administrator and the remote administration interface. In the proposed solution, the aforementioned vulnerability of the traditional scheme is addressed by reversing the client-server relation; an administration engine replaces the classical service, originating connections to an intermediate system instead of listening for connections.

The immediate advantage arising from this design choice is that there is nothing to attack on the remote host. On the other hand, the introduction of an additional system in the security chain must be carefully evaluated, to avoid introducing unexpected attack paths, and eventually making the system less robust than it originally was.

We claim that, if properly modeled and implemented, a platform based on the meeting of the server and its administrator on an intermediate system is expedient in terms of security, availability, usability and opportunity for future extension. This paper illustrates the architectural details of the proposed model, analyzes the implementation choices allowing to satisfy the security, efficiency and effectiveness constraints, and discusses the experimental validation of the resulting system.

2. Design concepts

A remote shell (see for example SSH [1]) is the most common tool for system administration. The concept is simple, yet perfectly fit for the intended usage: through a client-server protocol, a remote administration station accesses the same interpreter locally used on the target system to impart commands and show their results. However, the limitations deriving, on the one hand, from the inflexibility of the communication protocol, and on the other hand from the imperative nature of the shell, make this choice far from optimal. In the following, we will outline a quite ample general framework for the development of an alternative model, subsequently focusing on the solution of a particular issue.

2.1 The vision

The ultimate goal of this line of research is that of modeling an autonomous agent, able to receive external stimulations through a variety of channels, to judge the context surrounding it, and eventually to process the requests expressed by the administrators in the most appropriate fashion. Just like a real assistant, it should be able to accept directions through the medium most suitable to the peculiar context instead of being bound to a rigid communications protocol, and to network with other similar entities to pursue the goal of the optimal overall system configuration.

Within the A&A reference model [2], it is possible to foresee RoboAdmin as an autonomous computational entity, capable of self-determination; a proactive component of a complex network, seeing the actual targets of its management actions as artifacts [3]. The implementation of such an abstract and comprehensive concept, in our opinion, is within the reach of the innovative yet well-established coordination paradigms and technologies [4,5,6]. In this work, however, we will focus on a more limited goal: the modeling and practical implementation of one aspect of the illustrated vision, that is the substitution of the classical communication model with a more flexible and secure one.

2.2 The communication between the server and its administrator

The cited client-server protocol relies on the server part of the system to listen on a known network port, which becomes the obvious target for attacks targeting either protocol vulnerabilities or authentication weaknesses. Moreover, it often represents a critical single point of failure, taking the system completely out of control in case of problems.

The key concept of the proposed solution is abstracting the concept of remote management port, with the aim of attaining both greater usage flexibility and higher security. Instead of having a remote shell daemon passively listening for incoming connections from the management stations, RoboAdmin is based on an active management component, which connects to a meeting place (MP) where the administrator can contact it. For this approach to be effective and efficient, the MP has to satisfy the following properties:

I. not being an obvious entry point for the remote system
II. being easily reachable by the administrator
   - by employing a standard protocol
   - guaranteeing availability
III. supporting (possibly many layers of) authentication
IV. allowing the exchange of private information

Property (I) represents the key difference that the proposed model aims to achieve over the standard network service model. One of the simplest approaches to the problem of disguising the entry point for the remote system would be making the MP difficult to find, but this choice would affect both legitimate administrators and attackers; on the contrary, the abstract management port (AMP) has to be reachable at some predictable location, as stated by property (II). Our proposal is to pursue this goal by disguising the access point within a public, well-known MP, or better yet, within a set of MPs. Disseminating AMPs over several MPs (all reachable by means of the same, standard protocol) positively affects both the effectiveness of AMP camouflage and the availability of the service.

2.3 Existing technologies for building meeting places

The concept of a meeting place satisfying the aforementioned properties and allowing the instantiation of abstract management ports finds a quite natural mapping onto peer-to-peer communication networks and chat channels. These systems provide infrastructural support for the creation of places where people can meet. The name given to the places can change across different implementations, and the scheme used to reach them can vary from simple listings of places on a known server to complex, distributed location services, but the main features are essentially the same for most of the major platforms, i.e.:

- creation, publication and management of meeting places - the place is usually created on a specific host, and can be made visible to a global or local network through either a directory or a discovery system; it may be possible to transport the place over a different host during its lifetime; an access control mechanism can be available; administrative rights can be reserved to the creator of the place, shared or delegated;
- identity management - users accessing the place are identified by means of a nickname, which can either be independently chosen by them and disambiguated by the system in case of collision, or registered a priori; in the latter case, they can also be authenticated with a password; in any case, anonymity is the rule;
- public (broadcast) or private (one-to-one) communication - the main purpose of the place is usually letting every logged user see what others write, but users can also contact each other to start private conversations, restricted to a subset of the logged users.

2.4 Use (and abuse) of the meeting places by non-human players

While conceived for human-to-human communication, these systems have been the target of software agents, commonly known as bots, since their early history. The reasons for sending a bot in a meeting place are usually malicious [7,8]; they can listen to the conversation between the real users, trying to extrapolate sensitive data, or they can try to gain administrative rights for the meeting place. Research in the Artificial Intelligence field produced some interesting results in its quest to devise a software able to perform human-like conversation, thus passing the famous Turing test [9]. This goal has been pursued though different approaches, some exploiting clever but not really "intelligent" linguistic tricks [10], others building more complex AI-based system with learning abilities [11]. In the year 1994, Maudin coined the term chatterbot in its paper [12] describing the various conversational programs that appeared at the time.

The results are still quite rudimentary when compared against the ambition of convincingly mimicking a human conversation, yet they can be effectively fine-tuned for the simpler purpose of dissimulating the real nature of a bot.

In the end, RoboAdmin can take advantage of the technologies illustrated in this section and the preceding one, in order to implement an AMP in the form of a bot, deployed on a MP represented by a chat room or channel.

3. System architecture

According to the model described in the previous section, the logical architecture of RoboAdmin encompasses the four components depicted in Fig. 1. Their joint operation can be modeled as a illustrated in Fig. 2, and synthetically described as follows: a chatterbot on the server connects to the MP through a communication layer; when an administrator contacts the chatterbot and properly completes the authentication sequence enforced by an authentication module, the conversation is redirected to a command interpreter, which is the actual management interface of the server. In the following sections, the role and properties of each component is examined in detail.

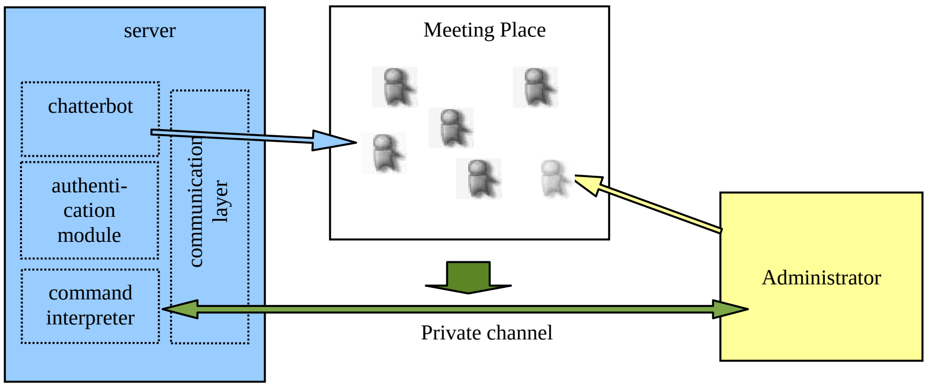

Fig. 1. RoboAdmin components and interactions with the MP

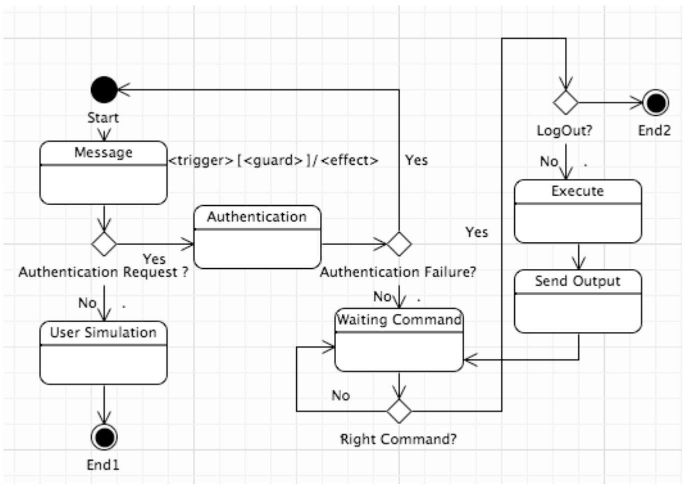

Fig. 2. Input processing sequence executed by the chatterbot

3.1 The chatterbot

The role of the chatterbot is establishing an access point to the management port within a meeting place. In order to satisfy the reachability property listed in section 2, the chatterbot connects to one or more MPs known to the administrator, using predefined identities.

Once an administrator has met the chatterbot acting as the access point for a remote server, he must provide proper authentication credentials in order to get access to the command interpreter. The chatterbot has to be programmed to recognize the context change, from the public section of the MP to a private communication channel, and when this happens it starts looking for authentication tokens within the conversation.

Since a legitimate administrator has obviously no interest in chatting, the bot can assume that a conversation beginning with any token different from an authentication request is coming either from an unsuspecting real user or from an attacker. Consequently, the chatterbot immediately leaves the pre-authentication state, and does not enter it again unless the private channel is closed and opened again.

A combination of factors allows to attain the desired security advantage over the classical, service-based approach:

- the chatterbot is programmed to behave as closely as possible like any human user; finding a RoboAdmin chatterbot in a densely populated MP represents an additional dimension in the search space for the potential brute-force attack.
- The server which sent the chatterbox out is unknown to the attacker. A real brute-force attack (that is, against an account protected by a strong password) is time-consuming and potentially traceable; it is usually driven by the motivation deriving from the knowledge of the target value, which RoboAdmin hides.
- Unlike a remote service, which very often is the one and only practical point of access to the server, RoboAdmin can send many chatterbots to different MPs, at the same time, or in sequence, either mimicking the typical activity cycles of real users or reacting to a problem that prevents correct operation on the current MP. The advantage is twofold:
  - the availability of the AMP is not tied to the correct working of a single MP.
  - Authentication failures can quickly trigger a locking-out reaction. In the classical scenario, locking out the system administrator for increasing time intervals (or even forever) after some failed login attempts is highly unpractical, and deeply affects the capability of the legitimate administrator of reaching the system, right at the time when quick intervention could potentially be most needed. Conversely, a chatterbot under attack can easily be disabled and replaced with another instance, possibly active on a totally different MP, with negligible impact on the legitimate administrator. In this scenario, brute-force attacks, even leveraging the knowledge of the authentication mechanisms by an adversary controlling the MP, can be very effectively thwarted.

3.2 The authentication module

In order to keep the chatterbot platform-independent, the only authentication-related function it performs is recognizing relevant tokens in the conversation and passing them to the authentication module. This component performs two functions:

- essentially, it acts as a gateway towards the platform-specific authentication infrastructure, like, for example, the system password verification library or an LDAP directory;

- it can implement any authentication scheme that can sensibly take place as a text-based, interactive protocol with a human administrator.

The locking-out policies and the consequent possible need for retracting a chatterbot from a MP and sending a new one out are configured within this component.

3.3 The communication layer

As explained, the communication model of RoboAdmin can be mapped onto any existing platform providing a few fundamental tools, namely:

- a public meeting place that can be unequivocally found on the Internet,
- support for establishing a private channel between users who met on the MP.

The specific protocol details are conveniently wrapped in a communication layer, providing an abstract view of the basic communication primitives (connection, message sending and receiving, etc.) to the chatterbot and the command interpreter, and encapsulating the advanced functions that can be found in some platforms, like, for example, authenticated access to the meeting places.

It is clearly possible to exploit different kinds of MPs at the same time, since the communication layer hides the specific details to the other components of RoboAdmin; the resulting AMP will exhibit an availability level greater than any simpler network service, not only due to its intrinsic redundancy, but also because it lowers the probability of seeing the administrator's client blocked by a firewall.

3.4 The command interpreter

After proper authentication, the command interpreter takes control of the interaction with the administrator. Since providing some out-of-band control mechanism would be uselessly cumbersome, this component must implement a logic for determining whether a sentence written by the administrator represents a command directed to the target system or intended for RoboAdmin. Its role, however, can reach well beyond this simple task, allowing to define a standard interface whose functions are translated into the appropriate actions on the target system. On platforms providing a complete text-based administration environment, both goals can be easily achieved by simply escaping shell commands, while on other platforms it may be necessary to provide a more complex conversion between the text commands issued by the administrator and calls to an administration API.

Eventually, both approaches can converge on a solution which presents a natural language interface to the administrator, performing an "intelligent" conversion of the abstract commands to whatever administration functions are found on the system. In this case, the command interpreter could be built on the same technology used for the chatterbot. Proposals for the implementation of a next-generation command-line interface based on Jabber have been made a few years ago [13].

4. Implementation and experimental validation

The illustrated architecture can be implemented using a wide variety of technologies, supporting almost any combination of target system (i.e. the one we want to remotely administrate) and network infrastructure suitable for building an AMP.

I order to build a simple prototype for the experimental validation of the concept, we chose to take advantage of the Internet Relay Chat (IRC) [14,15], an existing network architecture which has been proven successful for a task very similar to ours, albeit with very different intent: controlling infected hosts used for malicious purposes [16].

IRC was probably the first service allowing people to meet on the Internet and exchange text sentences in a many-to-many communications environment. It is still widely used, notwithstanding the large number of alternatives now available, and during its long history its has reached a very good level of stability and security. The protocol is well documented and client software is available in many flavors, often even on public-access network stations.

Regarding the target system, we implemented a command interpreter to wrap the bash shell on a Linux machine.

4.1 Operation

RoboAdmin connects to an IRC server and joins a channel. The communications layer component for this prototype can be easily configured to support both free-access and password-protected channels. The conversation flowing in the channel is continuously parsed by the chatterbot component, whose first task is determining whether other users are trying to establish an administration session or, unsuspecting of its nature, are simply approaching it to chat.

While the security of the system cannot rely on the effectiveness of RoboAdmin at disguising itself, it is useful to make its presence in a channel less obvious. For this purpose, the prototype exhibits a rudimentary intelligent behavior, implemented as a Prolog theory. In detail, we designed two sets of clauses on the TuProlog platform [5], the first one (d_c) coding "known questions" and the second one (d_n_c) coding "unknown questions", thus implementing an effective "change of subject" pattern structured as follows:

```
d_c([<expected_question>|T], [<related_answer>|O]):- !, d_c(T, O).
d_n_c([_], "<escape_sentence>").
```

where

<expected_question>	represents a string known to RoboAdmin
<related_answer>	represents a plausible follow-up to <expected_question>
<escape_sentence>	represents a plausible formula for terminating the conversation after the reception of an unrecognized sentence [_]

When the authentication module recognizes a valid attempt at switching to the command mode, and is able to correctly authenticate the administrator, the command interpreter takes control. It is implemented as a standard interpreter of the language defined by a completely configurable grammar, taking advantage of the existing tools for the Java language allowing to automatically derive the interpreter from the grammar definition. The following is an example of grammar defining some meta-commands directed at RoboAdmin; some of the commands can directly trigger the execution of the included shell code on the server:

```
<TOKEN>::=“REGISTER”|“IDENTIFICATION”|“IDENTIFY”|“EXIT”
          |“ABOUT”|“JOIN”|“SERVER&CHANNEL”|“REPEAT”|“EXECUTE”
<SH-C> ::= {<<shell_command>>}
<COMMAND> ::=!<TOKEN>! <SH-C>
```

4.2 Preliminary experimental results

The prototype has been running on a public host for six months, without incidents so far. It connects to a public IRC server, where the chatterbot actually got contacted by many other users, who rather quickly lost interest in its limited conversational ability, but apparently did not discover its real nature.

It is still too early for drawing conclusions, but at least it is possible to say that, until now, the prototype always correctly performed the job it was designed for.

5. Conclusions

The aim of this work was essentially that of starting a novel line of research regarding the problem of remote system administration, with an approach balancing theory and practice. For this reason, we worked on two parallel tracks: following the first track, we laid out a very high-level vision of what could be a framework for developing secure, autonomous agents assisting the system administrators in their efforts. Following the second track, we simplified the concepts of the high-level model enough to implement them in a working prototype, that could at least prove the feasibility of some solutions.

The result is a "bot" capable of disguising its presence within an IRC channel, and of acting as an intermediary between an administrator and a common Unix shell. The security of this realization of the Abstract Management Port concept relies on various factors: the difficulty of finding the bot and of associating it to a specific server, the (standard) difficulty of impersonating a legitimate administrator, and the much more effective anti-brute-force measures that can be put in place.

The main concerns, not yet addressed in the current implementation, regard passive authentication protocols. Of course it is not possible to use a passive authentication protocol within a public channel; the possibility of adopting human-computable active authentication protocols could overcome this problem, allowing to introduce additional security measures, the most notable being mutual authentication. In fact, a significant risk is posed by attackers mimicking a RoboAdmin chatterbot with the intent of stealing the authentication tokens from the administrator.

Among the functionalities that we plan to add in the short term, we deem very interesting the possibility of creating private group chat-rooms, where an administrator could manage a whole cluster of servers with a single command. In the long term, however, the main goal remains the definition of the theoretical framework, occasionally accompanied by experimental validation of specific components.

References

1. T. Ylonen, C. Lonvick, Ed. (S. Lehtinen): RFC 4250-54: The Secure Shell (SSH) Protocol series, http://www.rfc-editor.org/, January 2006
2. Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Coordination artifacts as first-class abstractions for MAS engineering: State of the research. In Alessandro F. Garcia, Ricardo Choren, Carlos Lucena, Paolo Giorgini, Tom Holvoet, and Alexander Romanovsky, editors, Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications, volume 3914 of LNAI, pages 71–90. Springer, April 2006. Invited Paper.
3. Ambra Molesini, Andrea Omicini, Enrico Denti, Alessandro Ricci: "SODA: a Roadmap to Artefacts" in Engineering Societies in the Agents World VI (O. Dikenelli, M. Gleizes, A. Ricci, Eds.), Lecture Notes in Artificial Intelligence, Vol. 3963, June 2006, 49--62, Springer
4. Andrea Omicini, Enrico Denti, Formal ReSpecT, in Declarative Programming - Selected Papers from AGP'00 (A. Dovier, M. Meo, A. Omicini, Eds.), Electronic Notes in Theoretical Computer Science, Vol. 48, 179--196, Elsevier Science B. V.
5. Enrico Denti, Andrea Omicini, Alessandro Ricci, tuProlog: a Light-weight Prolog for Internet Applications and Infrastructures, in Practical Aspects of Declarative Languages (I. Ramakrishnan, Editor), Lecture Notes in Computer Science, Vol. 1990, 184--198, Springer
6. A. Omicini, F. Zambonelli, Coordination of Mobile Information Agents in TuCSoN, Internet Research 8(5), 1998. MCB University Press.
7. Barnett, S. (2005). The IT crimewave. The Journal, 50(7), Lemos, R. (2003). Attack bot exploits Windows flaw. Retrieved Nov. 26, 2005, from News.com Web site: http://news.com.com/2100-1009-5059263.html.
8. Schluting, C. (2005). Botnets: Who Really "0wns" Your Computers?. Retrieved Nov. 26, 2005, from http://www.enterprisenetworkingplanet.com/netsecur/article.php/3504801.
9. A.M. Turing, "Computing Machinery and Intelligence," Mind, vol. LIX, no. 236, 1950, pp. 443-460.
10. A.L.I.C.E. (Artificial Linguistic Internet Computer Entity) - http://www.alicebot.org/
11. Jabberwacky - http://www.jabberwacky.com/
12. Michael L. Mauldin, Chatterbots, Tinymuds, And The Turing Test: Entering The Loebner Prize Competition, Proc. AAAI-94, Seattle, Washington
13. DJ Adams, Is Jabber's Chatbot the Command Line of the Future? - http://www.openp2p.com/pub/a/p2p/2002/01/11/jabber_bots.html
14. J. Oikarinen, D. Reed, Internet Relay Chat Protocol, RFC1459, May 1993 - http://www.rfc-editor.org/
15. C. Kalt, Internet Relay Chat series, RFC2810-13, April 2000 - http://www.rfc-editor.org/
16. Canavan, J. The Evolution of Malicious IRC Bots, White Paper, Symantec Security Response, 2005. - http://securityresponse.symantec.com/avcenter/reference/the.evolution.of.malicious.irc.bots.pdf