Scalable High-Assurance Technology for Detecting Compromised Host Computers

J. McDermott, W. Snook, and J. Luo
Information Technology Division

Introduction: Explosive improvements in commercial computer technology and tight budget constraints have driven the military to use commodity operating systems as the foundation for its latest information technology. Unfortunately, the ability of commodity operating systems to protect themselves has not greatly improved. (Commodity systems are defined as those that are both widely available and widely used.) As a result, it is difficult to build an information system that can protect military data according to its criticality and value. Additional technology is needed to strengthen the security infrastructure of military information systems. The problem we are addressing here is not one of nuisance attacks such as the viruses reported by the news media.

Deceptive Interpreters: For military and other national security systems, the problem we are concerned with is deceptive interpretation. The problem of deceptive interpretation occurs when some portion of a computer system is modified to present a false picture of reality.[1] A deceptive interpreter is a malicious agent that is capable of observing and changing the results of computations on its host system, according to a predefined strategy. Deceptive interpreters are able to change inputs for commands, the sequence of commands executed, or the information returned by computation in a way such that their policies are enforced.

Deceptive interpreters are analogous to the reference monitor concept for access control. The difference is that reference monitors are beneficial security mechanisms that enforce legitimate policies on the system. Deceptive interpreters, on the other hand, enforce malicious policies (which we call strategies) to the detriment of the host system. Another difference is that a deceptive interpreter must not be detected. A necessary strategy for any deceptive interpreter is that the deceptive interpreter itself must be hidden. Reference monitors can simply adopt fail-stop policies to exert control over systems; any request that violates the policy can be rejected and reported. Deceptive interpreters, on the other hand, cannot reveal their presence and must provide responses that appear legitimate. The originator of the legitimate commands must be misled without realizing it.

Rootkits: The most common use of deceptive interpreters is in association with a rootkit. After attackers gain root or administrative access to the system, they can install malicious tools including backdoors, sniffers, and tools to cover their tracks. These tools will run with root privilege and have the ability to fully control the system. However, backdoors and sniffers by themselves tend to have large signatures that could be easily detected. What makes rootkits exceptionally dangerous is the incorporation of deceptive interpreters that hide their presence. Deceptive interpretation can fool both automated tools and human system administrators into thinking their systems are safe. They enable a rootkit and its malicious payload to operate for an extended period of time, thus drastically prolonging the system compromise and escalating the damage.

The constant stream of new security vulnerabilities demonstrates that much of our technology is exploitable and at risk from deceptive interpretation. To inject deceptive interpretation into a military information system, it is only necessary to tamper with one link in the entire chain of computation (Fig. 1); preventing deceptive interpretation requires every link to be made tamper-proof. On the other hand, the effort for detecting deceptive interpretation is somewhere in the middle of those two extremes. Successful detection depends on monitoring the link that gets tampered with and recognizing that the tampering has occurred. The fundamental consequence of deceptive interpretation is that the host can no longer be trusted to inspect itself. A new technology is needed.

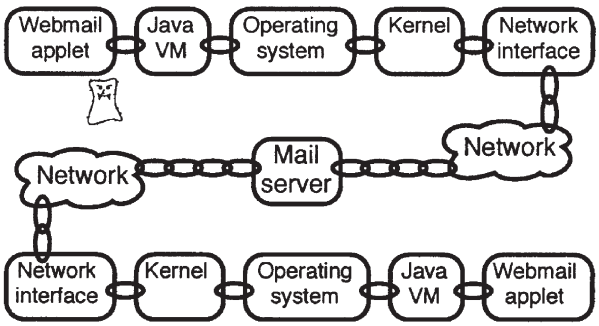

FIGURE 1
Tampering with the chain of computation.

Secure Attention Instruction: NRL researchers have focused on tamper-resistant hardware, software, and cryptographically protected means of detecting unauthorized modifications to commodity operating systems. The secure attention instruction concept provides a scalable infrastructure for detecting and report-

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2005		**00-00-2005 to 00-00-2005**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Scalable High-Assurance Technology for Detecting Compromised Host Computers	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Information Technology Division,4555 Overlook Avenue SW,Washington,DC,20375**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**4**	

ing compromised network hosts. This will provide a high-assurance security foundation for information-grid-based military systems.

A secure attention instruction can be a machine language operation code, an interpreted byte code, or a system call. In any of these forms, the effect is to cause the secure attention instruction (SAI) processor to perform a safety check of the host computer's software. The results of this check are signed by the SAI processor, using a cryptographic key that is protected inside the SAI processor. The results returned by the instruction cannot be used on the host computer that is protected by the secure attention instruction; a deceptive interpreter could tamper with that use. However, the results of a secure attention instruction can be checked on another host that is known to be secure. If this trusted host (called a mint) has the proper keys, it can confirm and report the authenticity of the results. If the safety check fails, security measures can be invoked to isolate, remove, or otherwise deal with the deceptive interpreter. If the mint sets a timer for each secure attention instruction that it creates, it can use the corresponding time-out event as a notification that the instruction may have been discarded by a deceptive interpreter. Security measures may be invoked on this time-out. Finally, since the secure attention instruction processor protects the keys it uses to authenticate the results, the deceptive interpreter cannot forge a result.

Figure 2 illustrates most of the key concepts needed to effectively construct a scalable secure attention instruction infrastructure. Part (a) shows all the components needed to use secure attention instructions in a simple network having a hierarchy of security domains. Part (b) is loosely structured as a unified modeling language (UML) collaboration diagram; it lists all the interactions involved in executing one secure attention instruction.

We begin by describing the objects shown in Fig. 2(a) moving clockwise from Mint P at the top. Mint P is a mint because it can create secure attention instructions that are a bit like currency in the sense that they are hard to forge. Because the figure includes three mints, we give each mint a name corresponding to its security domain; "P" is the label for the root domain of the entire network. The key in each mint represents the cryptographic material that allows the mint to create authenticatable and secret secure attention instructions. In the context of this section, we define a secure attention instruction as the string of bits that results when a mint applies a cryptographic function to a command such as x, y, or z. In the future, we plan to publish a description of the cryptographic exchange between a mint and an SAI device, but for now we assume that the protocol has already been established. Each mint also implements a policy of some sort to regulate the creation of secure attention instructions.

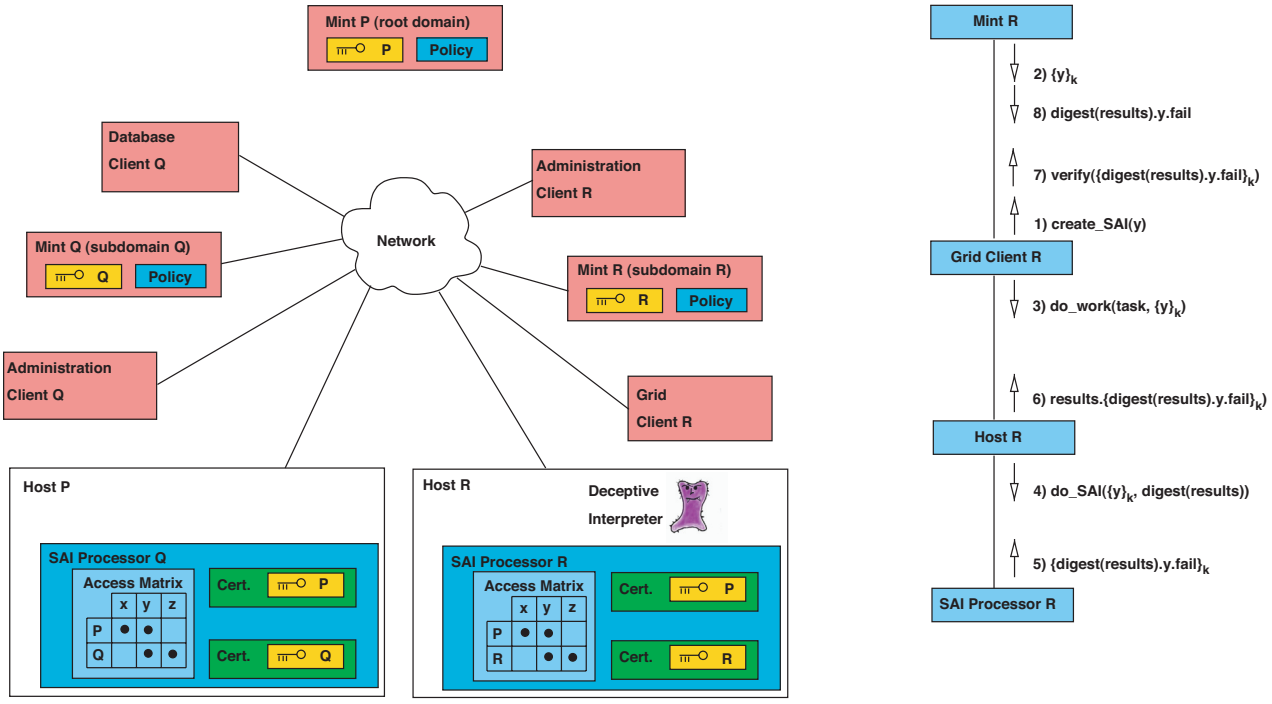

FIGURE 2
Key concepts for secure attention instructions.

Because Mint P is the root mint, it is not connected to the network, and its policy is to only create secure attention instructions for a small number of root domain administrators. The root mint does not create secure attention instructions often. When it does, the instructions contain commands for key management and other sensitive operations. We are making the assumption that the mint is trusted, and that it uses one of the many existing mechanisms to authenticate the administrators.

Next, Administration Client R represents the process (or computer) that the administrator for subdomain R uses to monitor and configure SAI processors within subdomain R.

Mint R has a different domain and policy than Mint P, but it is otherwise very similar. The policy of Mint R allows it to create two types of secure attention instructions, distinguished according to their commands. The first type, created only for subdomain administrators, is used to configure SAI processors. The second type, created for clients in the subdomain, implements the true purpose of the secure attention instruction. It allows clients to bind the results of some task performed on a remote computer to the results of a deceptive interpretation scan performed on the remote computer's SAI processor. The other important thing about the subdomain mint policy is that can include a timer for each secure attention instruction.

Grid Client R represents a process that a normal member of subdomain R uses to access grid computing services. Host R is a computer containing an SAI processor and running some sort of network daemon (such as Globus Toolkit). The computer is configured with a device driver and other appropriate software so that the daemon can interpret secure attention instructions by passing them on to the the SAI processor. Also, the icon with "fangs" represents a deceptive interpreter that has taken over a portion of the operating system on Host R in order to alter the results produced by the daemon. The SAI processor within Host R contains three interesting items. First, it has a "Cert," or certificate, for domain P and another for subdomain R. The certificates contain cryptographic material and other information allowing the SAI processor to authenticate and decrypt secure attention instructions minted by Mint P or Mint R. The third item is an access matrix that the SAI processor consults before carrying out commands. According to the matrix, SAI Processor R will honor commands x and y from Mint P, honor commands y and z from Mint R, and refuse all other commands. After the the SAI processor executes a command, it will typically return a result that can be authenticated and encrypted by the mint that issued the command.

Host Q, Administration Client Q, and Mint Q are configured for subdomain Q; otherwise, they are the same as their counterparts in subdomain R. Database Client Q is used for accessing a database service; otherwise, it is the counterpart of Grid Client R.

Figure 2(b) shows the basic exchange involved in executing a secure attention instruction. As mentioned earlier, the figure is loosely based on a UML collaboration diagram. The labeled boxes correspond to items in part (a) of the figure, and the lines between the boxes indicate collaboration. The numbered lines of text represent messages with semantics similar to a function call. The messages with parameters enclosed in "()" are like function calls, and the other messages are like return values. Each message starts with an arrow that points to its destination, and the messages are placed near their origin.

The secure attention instruction exchange proceeds in the following way. Grid Client R begins by asking Mint R to create an SAI containing the command y. (In an actual protocol implementation, the intended SAI processor should be specified here. Many other precautions would also need to be taken, so we use this simplification here for the sake of clear notation.) Mint R then responds with the SAI as requested. Next, in message 3, the client instructs Host R to perform a task and execute the SAI for y when it is finished. Host Device R performs the task and produces a result. Because the task specification included a description of the semantics for y, the host device took a digest of its results and passed them with the SAI to the SAI processor (message 4). The SAI processor performed command y which detected the presence of a deceptive interpreter in Host R. The processor then cryptographically bound its results to the host's digest and dispatched message 5. Next, the host appended the SAI result to its full task result and sent message 6 back to the client. The client extracted the SAI result and asked the mint to verify it in message 7. Upon receiving the message, the mint stopped the corresponding timer, noted the deceptive interpreter, and returned the digest and the result of y to the client. Finally, the client recalculated the digest and considered the result of the SAI. By using the SAI infrastructure, the client determined that the results were generated on the computer containing SAI Processor R and that they were not trustworthy because a deceptive interpreter was present.

NRL's proof-of-concept prototype will demonstrate the feasibility of integrating secure attention

instructions with host safety checks or scans. It will also prototype the scalable infrastructure needed to mint, confirm, and manage secure attention instructions.

Because we have limited resources, we chose to implement our prototype as a PCI-bus peripheral protecting the open-source Linux kernel. Intelligent peripherals have the necessary local processing power and interface restriction capabilities to be relatively omniscient (as bus masters) but tamper-resistant. An Intel IQ80310 prototyping board provides these features and saves hardware prototyping effort.

[Sponsored by ONR]

References

[1] G. Karjoth and J. Posegga, "Mobile Agents and Telcos Nightmares," *Annales Telecommun*, **55**(7/8), 29-41 (2000).