Simulation of Computer Network Attacks

Carlos Sarraute[1,2], Fernando Miranda[1], and Jose I. Orlicki[1,2]

[1] CoreLabs, Core Security Technologies
[2] ITBA (Instituto Tecnológico de Buenos Aires)

Abstract. In this work we present a prototype for simulating computer network attacks. Our objective is to simulate large networks (thousands of hosts, with applications and vulnerabilities) while remaining realistic from the attacker's point of view. The foundation for the simulator is a model of computer intrusions, based on the analysis of real world attacks. In particular we show how to interpret vulnerabilities and exploits as communication channels. This conceptual model gives a tool to describe the targets, actions and assets involved in multistep network attacks. We conclude with applications of the attack simulator.

1 Introduction

We present a network simulator focused on the attacker's point of view, based on a model of network attacks. We begin with a brief description of real world attacks, and describe an abstraction of the attack actions (section 2). We delve in more detail in the "Attack and Penetrate" phase of an attack, in which the attacker exploits a vulnerability to gain access to a target machine (section 3). Again, we show how to abstract and generalize the process of exploitation and payload execution. Then we present the prototype for simulating network attacks, based on the implementation of multiplatform agents and the abstraction of vulnerabilities and exploits as communication channels (section 4). The next section deals with the tension between realism and performance in the simulation. Finally we mention some applications of the simulator.

2 From Real World Attacks to an Attack Model

2.1 Computer Network Intrusions

During a network intrusion, an attacker tries to gain access to software systems that require authorization. The intrusion may be illegal, or may be an authorized audit performed by security professionals. The latter is called a network penetration test: an authorized attempt to compromise network security and access sensitive information by taking advantage of vulnerabilities. As networks evolve, and combine a multitude of interconnected technologies, the penetration test has become an accepted practice to evaluate the global security of a network (ultimately assessing effectiveness of the deployed security countermeasures). Since pentesters basically use the same tools and methodologies as unauthorized attackers, we can focus on the former (whose practices are also more documented!)

2.2 Main Steps of an Attack

1. **Information Gathering.** A successful attack depends on the ability to gather relevant information about the target network, including active IP addresses, operating systems and available services. Actions realized during this phase include:
   - Network discovery: performed utilizing mechanisms including ARP, TCP SYN packets, ICMP echo request, TCP connect and passive discovery.
   - Port scanning: an exhaustive scan of open and closed ports of all the network hosts.
   - OS identification: consists of recognizing the OS of a remote host by analyzing its responses to a set of tests. Classical Nmap's fingerprinting database can be combined with a neural network to accurately match OS responses to signatures, see [5]. Additional OS identification capabilities are available for more specific situations. For instance, OS detection utilizing the DCE-RPC and SMB protocols can identify Windows machines more precisely.
2. **Attack and Penetrate.** During this phase, the attacker selects and launches remote exploits making use of data obtained in the Information Gathering step. As we will see in section 3, an exploit can be thought of as a way to install an agent on a compromised host.
3. **Local Information Gathering.** The attacker collects information about computers that he has successfully compromised: OS, network configuration, users and installed applications, rights obtained on the system.
4. **Privilege Escalation.** The attacker attempts to obtain administrative privileges (to gain root or superuser privileges) by running local exploits.
5. **Pivoting.** After Privilege Escalation, the attacker can use the newly controlled host as a vantage point from which to run attacks deeper into the network. By sending instructions to an installed agent, the attacker can run local exploits to attack systems internally, rather than from across the network. He can view the networks to which a compromised computer is connected, and launch attacks to other computers on the same network, gaining access to systems with increasing levels of security. That is, the attacker executes the previous steps (Information Gathering and Attacking) using the new agent as a source.
6. **Clean Up.** The attackers needs to clean up his steps to avoid detection. Towards this end, all the executed actions should minimize the noise produced, for example, by making modifications only in memory and by not writing files in the target's filesystem.

2.3 Abstraction of Attack Actions

After this brief review of the steps of a real world attack, we present here the model that we use as abstraction of an attack. The conceptual building blocks are Assets, Actions and Agents.

Assets. An asset can represent anything that an attacker may need to obtain during the course of an attack. More precisely, it represents the knowledge that an attacker has of a real object or property of the network. Examples of assets include:

```
* BannerAsset (banner, host, port)
* OperatingSystemAsset (os, host)
* IPConnectivityAsset (source, target)
* TCPConnectivityAsset (source, target, port)
```

A BannerAsset represents the `banner` that an attacker obtains when trying to connect to a certain `port` on a `host`. An OperatingSystemAsset represents the knowledge that an attacker has about the operating system of a `host`. A TCP-ConnectivityAsset represents the fact that an attacker is able to establish a TCP connection between a `source` host and a certain `port` of a `target` host.

The assets we consider are probabilistic. For example, an action which determines the OS of a host using banners (OSDetectByBannerGrabber) may give as result an OperatingSystemAsset `os=linux` with `probability=0.8` and a second one with `os=openbsd` and `probability=0.2`.

Attack Actions. These are the basic steps which form an attack. Examples of actions are: Apache Chunked Encoding Exploit, WuFTP globbing Exploit (subclasses of Exploit), Banner Graber, OS Detect by Banner, OS Fingerprint, Network Discovery, IP Connect and TCP Connect. We review below the principal attributes of an action.

1. **Action goal.** When executed successfully the action completes the asset associated with its goal (also called the *action result*). It is common to speak about the result of an action focusing on non authorized results (for example to increase access, obtain information, corrupt information, gain use of resources, denial of service). This is a particular case of our concept of goal, since we also consider different types of goals like gathering information or establishing connectivity between two agents or hosts, that allow us to model the intermediate steps of an attack.
2. **Action requirements.** The *requirements* are goals of other attack actions, that must have been successfully executed before the considered action can be run. These relations can be used to construct an attack graph. By analyzing of the attack graph, the attacker can build a plan (as a sequence of actions) to reach the final objective. On the use of attack graphs for automated planning, we refer the reader to [8].
3. **Environment conditions.** The environment conditions refer to system configuration or environment properties which may be necessary or may facilitate the execution of the action. For example, the FTPD Glob Overflow Exploit, which exploits a buffer overflow and runs only on specific versions of OpenBSD, includes as environment condition “OS=OpenBSD; version between 2.7 and 2.8”. These conditions are not necessary, but influence the probability of success and the result of the action.

4. **Noise produced and stealthiness.** The execution of the action will produce *noise*. This noise can be network traffic, log lines in IDS, etc. Given a list as complete as possible of network sensors, we quantify the noise produced respective to each of this sensors. The knowledge of the network configuration and which sensors are likely to be active, allows us to calculate a global estimate of the noise produced by the action. The *stealthiness* of an action refers to the low level of noise produced.
5. **Running time and probability of success.** The *expected running time* and *probability of success* depends on the nature of the action, but also on the environment conditions, so their values are updated every time the attacker receives new information about the environment. These values are necessary to take decisions and choose a path in the graph of possible actions. Together with the stealthiness and zero-dayness, these values constitute the cost of the action and are used to evaluate sequences of actions.

3 On Vulnerabilities and Exploits

3.1 Anatomy of an Exploit

The exploits are the most important actions during an attack. An exploit is a piece of code that attempts to compromise a workstation or desktop via a specific vulnerability. According to the literal meaning of exploit, it takes advantage and makes use of a hidden functionality. When used for actual network attacks, exploits execute payloads of code that can alter, destroy or expose information assets (see [3] for a survey on exploit development).

Exploited Vulnerability. To obtain an unauthorized result, the exploit makes use of a vulnerability. This can be a network configuration vulnerability, or a software vulnerability: a design flaw or an implementation flaw (buffer overflow, format string, race condition).

The most classic example is the buffer overflow, first described in "Smashing the stack for fun and profit" by Aleph One (1996). Questions that the exploit writer must solve include how to insert code and how to modify the execution flow to run it. In the example of a stack based buffer overflow, the code is inserted in a stack buffer and by overflowing the buffer, the attacker can overwrite the return address and jump to his code.

Payload. Once he manages to trigger and exploit a bug, the attacker gains control of the vulnerable program. The payload is the functional component of the exploit, the code that the attacker is interested in running. The most popular payloads are called *shellcode*. This technique consists in opening a shell (a command interpreter), that the attacker can use to execute available commands.

Writing payloads is a very difficult task, that requires solving multiple constraints simultaneously. The payload is a sequence of byte codes, so each payload will only work in a specific operating system and platform. Depending on the

attack vector, the payload may be sent to the vulnerable machine as an ASCII string (or some protocol field), and thus must respect a particular grammar (examples: byte 0 is forbidden, only 7-bit ASCII is accepted, only alphanumeric characters are accepted, etc.) Libraries have been developed to help exploit writers to generate shellcodes. The open source libraries "MOSDEF" [1] and "InlineEgg" [13] are two well known cases, with tools to cope with the restrictions. The payload is also typically limited in size (for example the buffer size in the case of a buffer overflow), so the code that the attacker will run must fit in a few hundred bytes. If he wants to execute more complex applications, he must find another way.

3.2 Universal Payload

We present here the solution called "syscall proxy" (developed by Max Caceres et al., see [6] for more details). The idea is to build a sort of "Universal Payload" that allows an attacker to execute any system call on the vulnerable host. By installing a small payload (a thin syscall server), the attacker will be able to execute on his local host complex applications (a fat client), with all system calls executed remotely.

Syscalls. A software process usually interacts with certain resources: a file in disk, the screen, a networking card, a printer, etc. Processes can access these resources through *system calls* (*syscalls* for short). These syscalls are operating system services, usually identified with the lowest layer of communication between a user mode process and the OS kernel. For example, a process that reads data from a file might do so using the open, read and close syscalls.

Syscall proxy. Syscall proxying inserts two additional layers between the process and the underlying operating system. These layers are the *syscall client* layer and the *syscall server* layer (see figure 1).

The syscall client layer (which runs on the attacker's machine) acts as a link between the running process and the system services on a remote host. This layer is responsible for forwarding each syscall argument and generating a proper request that the syscall server can understand. It is also responsible for sending this request to the syscall server, usually through the Internet, and returning back the results to the calling process.

The syscall server layer (which runs on the compromised system) receives requests from the syscall client to execute specific syscalls using the underlying operating system services. After the syscall finishes, its results are marshaled and sent back to the client, again through the Internet.

There are multiple connection methods between agents. The attacker can chose: connect to target (the new agent listens for incoming connections on a specified network port and protocol, usually TCP), connect from target (the new agent initiates the connection from the compromised system), reuse socket (reuses the connection method used to trigger the vulnerability, thus making

use of a communication channel that the attacker knows to work) and HTTP tunneling. Agents can also be chained together to reach network resources with limited connectivity.

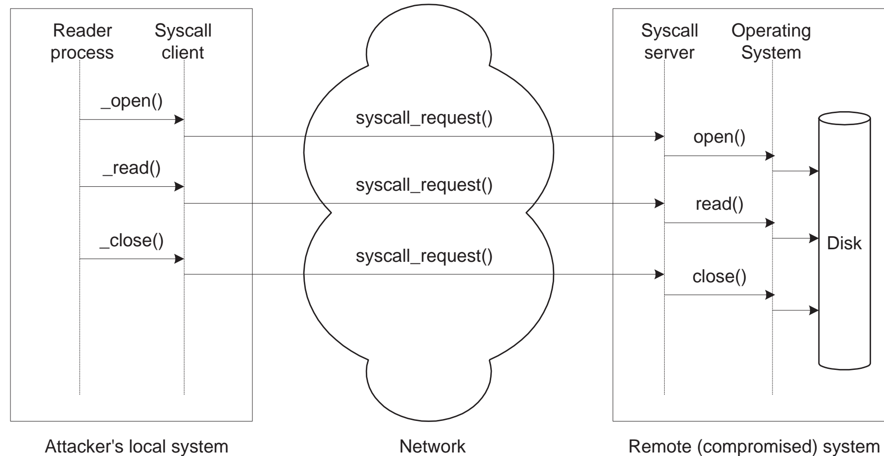

Fig. 1. Diagram of a Proxy Call Execution

3.3 Attack Agents

In our attack model, the abstraction of arbitrary syscall servers is the concept of Agents. The Agents are responsible for executing the attack actions. Thus an attack typically begins with a local agent (representing the attacker, which can interact with the local agent through a graphical console), and follows with the main steps of an attack (as described in 2.2), where the successful exploitation of a vulnerability means installing an agent.

The attacker is modeled as a set of connected agents, and exploits are modeled as probabilistic actions that depend on the details of target OS and applications. Thus, we can simulate the interactions between an attacker and the network by simulating the behavior of the agents.

4 Large Network Simulator

4.1 Focus on the attacker's point of view

We now present our implementation of a network simulator specifically designed to simulate network attacks. Our objective is to simulate very large networks, for example 2.000 machines simulated on a single desktop PC. It is of course not

feasible to simulate all the network traffic, or to use a VMware server running simultaneously 2.000 images.

The idea of our implementation is to focus on the point of view of the attacker. Using our model of the attacker, we can build a simulator which is realistic from his point of view. The simulator is interactive, and should be accessed through an application that actively attacks a real network (such as Core Impact [7]). The output of the simulator is to respond in real time to syscall requests. In particular, the simulator only generates information as requested by the attacker. By performing this lazy evaluation, the main performance bottleneck comes from the ability of the attacker to request information from the network.

4.2 What scenarios can be simulated?

The simulated scenarios are composed of machines, networking devices and vulnerabilities. Supported machine components are: windows workstations and servers, many Unix systems, routers, proxies, firewalls and Intrusion Detection Systems. Each machine can be independently configured, and installed to run different software services, such as Web server, FTP server, telnet and secure shell. New applications can be developed for the simulator platform using the usual development tools. Network components are used to interconnect machines, and can simulate hubs, switches, vlans and dialup connections and their security characteristics. Vulnerability descriptions are entered in the vulnerabilities database, to allow the simulation of the vulnerable application behavior.

4.3 Multiplatform Agents

According to our attack model, an attacker can be effectively modeled by a set of agents. Thus by simulating the behavior of the agents, we can simulate the behavior of the network (this is transparent for the attacker). The whole environment is accessed by the attacker through the local agent, and interactions take place in the form of proxied system calls.

The base of the simulator are agents that responds proxycalls. The agents implement a syscall server for a specific SO and platform, but all of them have the same interface: this is what we call “Multiplatform Syscall”. So, if an attacker (client) can install a multiplatform syscall agent in a victim host, he does not care about the syscalls supported by the target host, the attacker only needs to know the universal syscall interface exported by the agent.

4.4 The Semantics of the Exploit Database

Security Model. In the simulator security model, a *vulnerability* is a mechanism used to (potentially) access an otherwise restricted communication channel. An *exploit* is a “magic” string that opens access to some vulnerable agent’s channel. It can be simulated as a message with a symbolic identifier, sent to an application. Depending on the environment conditions, the exploit database will determine the resulting behavior of the application.

Given a target machine M, the simulator iterates the list-like structure of *results* in order. Each result entry has conditions associated to it, so the simulator iterates the tree-like structure of *requirements* section and, if a match is found, the action (install an agent, crash or reset) is executed with probabilistic behavior. The execution of actions stops when an action is evaluated to True.

Requirements. In the *requirements* section, you can use several kind of tags. The tags specify the conditions that have influence on the execution of the exploit (that is on the result probabilities). Example:

```
<requirement type="system" id="req0">
  <os arch="i386" name="windows" />
  <win>nt4</win>
  <edition>server enterprise_server</edition>
  <servicepack>6 6a</servicepack>
</requirement>
```

This states that one of the possibilities is that the target machine runs Windows version NT4, the edition should be "server" or "enterprise_server" and the service pack should be 6 or 6a. The requirements have a unique id to identify them, in this case "req0". Another requirement concerns the target application:

```
<requirement type="application" id="req1">
  <status>target</status>
  <name>Internet Information Services</name>
  <version major="4 5" />
</requirement>
```

This states that the machine should be running "Internet Information Services", version major 4 or 5, and this application is the target of the exploit.

The possible status are:
1. *target*: the application is the target of the exploit (the most common case).
2. *running*: the application should be running but is not necessarily the target of the attack.
3. *installed*: the application should be installed but not necessarily running.
4. *not running*: for example, a remote exploit will have more success probability if the target machine is in a network with no firewalls running.

Requirements can be combined, for example:

```
<requirement type="compose" id="req2">
  <operator>logic_and</operator>
  <operands>req0 req1</operands>
</requirement>
```

The result of the "logic_and" operation is a requirement stating that the target machine should be running Windows NT4 server edition or enterprise_server edition, and running IIS (Internet Information Services). There is also a "logic_or".

Results. The result is a list of the relevant probabilities, for example:

```
<result for="req1">
  <crash chance="0.00" what="os" />
  <reset chance="0.00" what="os" />
  <crash chance="0.10" what="application" />
  <reset chance="0.00" what="application" />
  <agent chance="0.75" />
</result>
```

In order, these are: the chance of crashing the machine, of resetting the machine (reboot), of crashing the target application (IIS), of resetting the target application, and of successfully installing an agent.

To determine the result, we follow this procedure: processing the lines in order, for each positive probability, choose a random value between 0 and 1. If the value is smaller than the chance attribute, the corresponding action is the result of the exploit.

In this example, we draw a random number to see if the application crashes. If the value is smaller that 0.10, the application IIS is crashed and the execution of the exploit is finished. Otherwise, we draw a second number to see if an agent is installed. If the value is smaller than 0.75, the agent is installed, otherwise there is no visible result.

5 Performance Issues

5.1 Simulation versus emulation

From a systemic view-point, we speak of simulation when the level of detail of interaction between components inside the system is mimicked, and emulation when only the interaction of the system with the environment is mimicked. Following this line of thought, the system implemented simulates networks in the socket abstraction level, and inside the network the behavior of machines is emulated from the communication angle. The emulation of computers is basic but complete, in the sense that a remote virtual user connecting to one of them can execute different processes and handle data files.

5.2 Tension between realism and performance

There is a tension between realism and performance in the simulation. In this case, good performance is achieved by only simulating the syscalls / socket abstraction level. Most actions work at the syscall level and attack upper levels of abstraction, whereas the network packet switching is not simulated.

The network simulator was designed to be able to simulate networks of thousands of computers. Each simulated machine has at least one thread. The goal was to have a simulator on a single desktop computer with a simulated traffic

realistic from a penetration test point of view. It was not designed to simulate DDoS (Distributed Denial of Service) attacks as floods or worms but there is a possibility in that direction also, maybe in dedicated servers running the simulation.

5.3 Socket direct

A hierarchy for file descriptors was developed, including a variety of sockets optimized for the simulation in one computer, called "socket direct". Socket direct is fast: as soon as a connection is established, the client keeps a filedescriptor pointing directly to the server's descriptor. Routing is only executed during the connection. Process Control Blocks (PCB) are created as expected, but are only used during connection establishment There is support for TCP and UDP sockets, and a central set of systems calls, including filesystem syscalls, to emulate memory in each machine of the network. Data enters to the simulation through the socket subclass "socket real", which wraps a real BSD socket of the underlying operating system.

5.4 Scheduler

The responsibility of the scheduler is to assign the CPU resources to the different machines in the simulation and inside each machine to the different processes. The scheduling is non preemptive and round-robin. The scheduling iterates over the hierarchy machine / process / thread as a tree (like a depth-first search). Simulated threads are real threads of the OS, simulated machines and processes are all running within the unique process of the simulator. Thanks to this architecture, there is no loss of performance due to context switching (descriptors and pointers remain valid when switching from one machine to the other).

Something to remark is that the simulator doesn't have to use all the CPU when idle, so the scheduler was devised to sleep (e.g. 20ms) the simulator after executing a constant number (e.g. 512) of machine runs (runs to sleep). This leaves space for other programs interacting with the simulator to continue their normal activity in the desktop machine while the simulator is idle.

Another improvement was to change the runs to sleep dynamically in a exponential increment and linear back-off fashion, depending on a threshold of syscalls lost per sleep. This results in better overall response when there is simulated activity and less use of the CPU when there is no simulation activity.

Figures 2 and 3 show measurements that were done on a Pentium D 2.66Ghz machine with 1.5GB of RAM, running Windows XP SP2. The simulated scenario includes 100 networks of 10 machines each, so there are 1.000 machines running in the simulation. When responding to a TCP Port Scan or an OS Detection by Banner Grabber, the simulator answers between 700 and 900 syscalls per second. Total running time of the modules (on a single network) lies between 100 and 120 seconds.

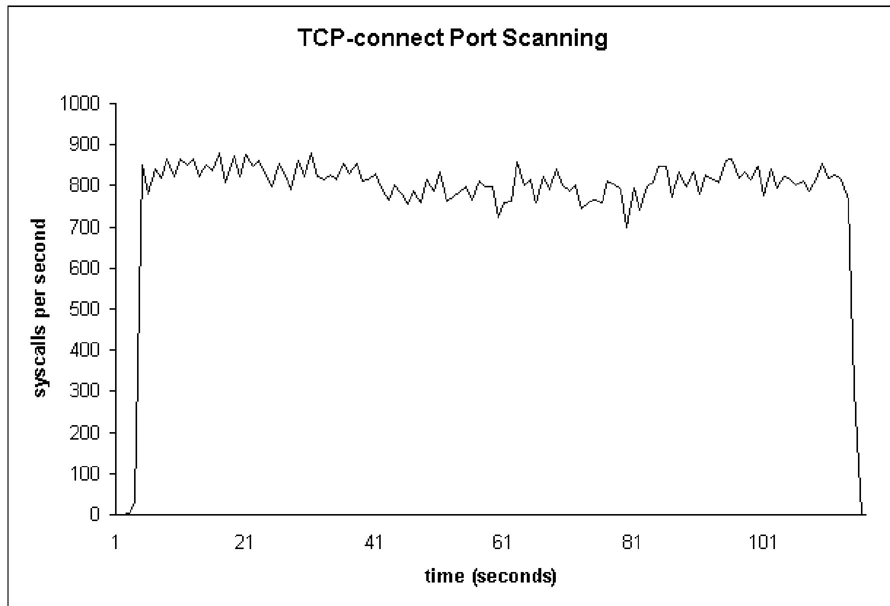

Fig. 2. Simulated TCP Port Scanning (time versus syscalls/second)

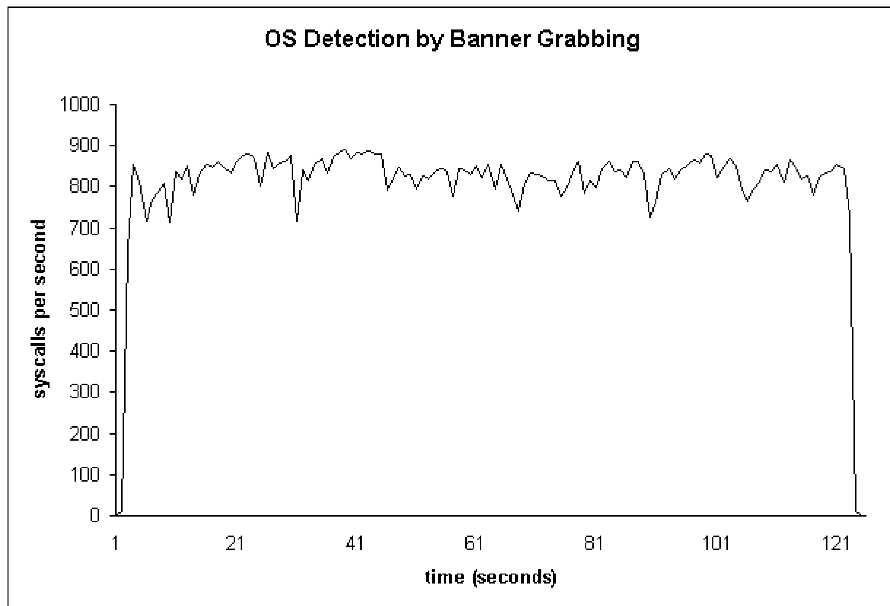

Fig. 3. Simulated OS Detection by Banner Grabber (time versus syscalls/second)

5.5 File system with templates

In order to handle thousand of files, avoiding wasting of huge disk space, the filesystem simulation is accomplished mounting template filesystems. A *template filesystem* is a common file repository shared by a group of virtual machines. For example, all Windows systems share a common Windows file repository with the default installation provided by Microsoft. These templates have read permission only, so when a machine needs to read or change a file, the file is copied to a local filesystem in that machine, this technique is well know as "copy on write". The fundamental idea is that if multiple callers ask for resources that are initially indistinguishable, you can give them pointers to the same resource. This function can be maintained until a caller tries to modify its copy of the resource, at which point a true private copy is created to prevent the changes from becoming visible to everyone else. All of this happens transparently to the callers. The primary advantage is that if a caller never makes any modification, no private copy needs to be created.

In order to improve the performance, a file cache was implemented: the simulator saves the most recent accessed files (or block of files) in memory. In high scale simulated scenarios, it is very common to have several machines doing the same task at (almost) the same time. For example, when the system starts up, all UNIX machines read the boot script from `/etc/initd` file; if these kind of files are in the system cache, the booting process is faster, because only few disk accesses are needed, even in scenarios of hundreds or thousands of simulated machines.

6 Conclusion

We presented a network simulator focused on the attacker's point of view. The simulation is based on a model of network attacks, whose building blocks are Assets, Actions and Agents. By making use of the proxy syscalls technology, and simulating multiplatform agents, we were able to implement a simulation that is both realistic and light-weight, allowing the simulation of networks with thousands of hosts. Some applications of the simulator are:

- Cyber attack modeling and analysis tool. The different security components can be configured to report attack evidence in the same way as the real world components, allowing, for example, post-attack forensics analysis and real-time detection exercises.
- Pentest training tool. A step by step tutorial for pentesters is hard to write because the user might not have a proper target network setting, or because the characteristics of the user's target network are unknown. The simulator can be used to deploy several complex scenarios in the user's computer, so that the user can follow the training on a shared scenario. This application has been tested with security professionals, both inside and outside the company, and got positive feedback.

– Evaluation of countermeasures. Consider a system administrator that has a set of measures which make certain attack actions less effective (in our framework, a measure may reduce the probability of success of an attack action, or increase the noise it produces, for example by adding a new IDS). He can then use the simulation to see if his system becomes safe after all the measures are deployed, or to find a minimal set of measures that make his system safe.

References

1. Dave Aitel, "MOSDEF library", `http://www.immunitysec.com/resources-freesoftware.shtml`
2. Aleph One, "Smashing The Stack For Fun And Profit", Phrack Magazine 7, 49 (1996).
3. Ivan Arce, "Attack Trends - The Shellcode Generation", IEEE Computer Society - Security & Privacy Magazine, Vol. 2, No. 5.
4. Avrim L. Blum and John C. Langford, "Probabilistic planning in the GraphPlan Framework", AIPS98 Workshop on Planning as Combinatorial Search, pages 8-12, June 1998.
5. Javier Burroni and Carlos Sarraute, "Outrepasser les limites des techniques classiques de Prise d'Empreintes grace aux Réseaux de Neurones", Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC), Rennes, France, May 31-June 2, 2006.
6. Max Caceres, "Syscall Proxying - Simulating remote execution", Black Hat USA 2002 Briefings and Training, July 29-August 1, 2002.
7. Core Security Technologies, "Core Impact", `http://www.coresecurity.com`
8. Ariel Futoransky, Luciano Notarfrancesco, Gerardo Richarte and Carlos Sarraute, "Building Computer Network Attacks", CoreLabs Technical Report, March 2003.
9. Somesh Jha, Oleg Sheyner, Jeannette Wing, "Minimization and Reliability Analyses of Attack Graphs", February 2002.
10. John D. Howard, Thomas A. Longstaff, "A Common Language for Computer Security Incidents", Sandia Report, October 1998.
11. Ulf Lindqvist, Erland Jonsson, "How to Systematically Classify Computer Security Intrusions", Proceedings of the 1997 IEEE Symposium on Security and Privacy, May 1997.
12. Andrew P. Moore, Robert J. Ellison, Richard C. Linger, "Attack Modeling for Information Security and Survivability", Software Engineering Institute Technical Report, 2001.
13. Gerardo Richarte, "InlineEgg library", `http://oss.coresecurity.com/projects/inlineegg.html`
14. Bruce Schneier, "Attack Trees: Modeling Security Threats", Dr. Dobb's Journal, December 1999.
15. Bruce Schneier, "Secrets and Lies: Digital Security in a Networked World", Chap. 21 Attack Trees, Wiley Computer Publishing, 2000.
16. Laura P. Swiler, Cynthia Phillips, Timothy Gaylor, "A Graph-Based Network-Vulnerability Analysis System", Sandia Report, January 1998.
17. T. Tidwell, R. Larson, K. Fitch, J. Hale, "Modeling Internet Attacks", Proceedings of the 2001 IEEE Workshop of Information Assurance and Security, June 2001.