9-2002

A Genealogical Approach to Analyzing Post-Mortem Denial of Service Attacks

Greg Rice
Iowa State University

James A. Davis
Iowa State University, davis@iastate.edu

A Genealogical Approach to Analyzing Post-Mortem Denial of Service Attacks

Abstract
Availability requires that computer systems remain functioning as expected without loss of resources to legitimate users. The impact of a lack of availability to services and data is often little more than a nuisance; however the results could be devastating if critical computational and communication resources are targeted. One of the most problematic challenges to availability is the denial of service (DoS) attack. Over time, DoS attacks have become increasingly sophisticated, often employing techniques like address spoofing, coordinated distributed sources of attack, and subverting "inside" computers to assist in carrying out the attack. DoS attacks are very easy to launch, are effective, and are difficult to prevent or mitigate.

The purpose of this work is to study post-mortem DoS attacks over time with the goals of uncovering how the attacks relate to each other, identifying the underlying vulnerability that led to success, and gaining insight on future attack trends. By studying how attacks have changed over time and adapted to overcome new security practices, it is possible to construct attack trees to represent the genealogy and history of DoS attack tools. Through code inspections and close analysis of the attack trees, we were able to identify core techniques copied from one attack to another, the synthesis of more effective techniques based on combinations of existing methods, and the genesis of novel attack strategies. The generation of attack trees allows for an important examination of how attacks relate to one another as well as insight on the core vulnerabilities that still remain in modern software solutions. More importantly, by closely analyzing the genealogy of attack trees and post-mortem DoS exploitation, we not only gain information on the methodologies currently used by attackers but also discover valuable insight on predicting future attack patterns as well as developing possible countermeasure.

Disciplines
Information Security | Systems and Communications

A Genealogical Approach to Analyzing Post-Mortem Denial of Service Attacks

Greg Rice and James Davis
Department of Electrical and Computer Engineering
Iowa State University
{grice, davis}@iastate.edu

ABSTRACT

Availability requires that computer systems remain functioning as expected without loss of resources to legitimate users. The impact of a lack of availability to services and data is often little more than a nuisance; however the results could be devastating if critical computational and communication resources are targeted. One of the most problematic challenges to availability is the denial of service (DoS) attack. Over time, DoS attacks have become increasingly sophisticated, often employing techniques like address spoofing, coordinated distributed sources of attack, and subverting "inside" computers to assist in carrying out the attack. DoS attacks are very easy to launch, are effective, and are difficult to prevent or mitigate.

The purpose of this work is to study *post-mortem* DoS attacks over time with the goals of uncovering how the attacks relate to each other, identifying the underlying vulnerability that led to success, and gaining insight on future attack trends. By studying how attacks have changed over time and adapted to overcome new security practices, it is possible to construct attack trees to represent the genealogy and history of DoS attack tools. Through code inspections and close analysis of the attack trees, we were able to identify core techniques copied from one attack to another, the synthesis of more effective techniques based on combinations of existing methods, and the genesis of novel attack strategies. The generation of attack trees allows for an important examination of how attacks relate to one another as well as insight on the core vulnerabilities that still remain in modern software solutions. More importantly, by closely analyzing the genealogy of attack trees and *post-mortem* DoS exploitation, we not only gain information on the methodologies currently used by attackers but also discover valuable insight on predicting future attack patterns as well as developing possible countermeasure.

BACKGROUND

In February of 2000, a series of massive attacks incapacitated several high-profile Internet sites, including Yahoo, Ebay, and E*trade. Next, in Januaryof 2001, Microsoft's name server infrastructure was disabled by another assault [1]. Since September 1996, several dozen sites on the Internet have been subjected to similar attacks [2]. The attacks can be launched with little effort and are often difficult to trace back to the originator. Furthermore, it is never necessary for the attacker to explicitly break security mechanisms to exploit the vulnerability. Complex and strong authentication mechanisms that provide data confidentiality and integrity are often rendered useless in availability attacks since the enemy never actually needs to compromise the computer system security.

Availability requires that computer systems remain functioning as expected without degradation in processing and that resources remain accessible to legitimate users. Although many organizations may have implemented well planned and good security practices in building their networks, these networks still remain open to common availability attacks. In other words, traditionally well-protected infrastructures suffer from vulnerabilities in maintaining availability. In fact the existence of previous security practices and management plans may indeed actually make the network more vulnerable to attack due to the false pretense of immunity.

The primary goal of an availability attack is simply to deny victims access to a particular computer resource. The enemy never needs to supply a password, secure token, or other means of authentication; the attacker only needs to block other users from accessing the system. A denial of service attack is characterized by an explicit attempt by attackers to prevent users of a service from using that service [3]. For example, flooding

a network with traffic prevents other computing systems from communicating on the same network and thereby disrupts normal network services. In this scenario the attacker simply overloads normal resources and effectively blocks communication.

Many of these assaults, otherwise known as denial of service (DoS) attacks, are commonly published on a wide variety of Internet web sites. In fact many sites freely advertise and distribute easy-to-use attack scripts. Web-surfers, who may know little of the technical details of the attack, can easily download the scripts and launch large bombardments against computer networks and computer systems. As a result, many attacks are easy to carry out and may even occur on a relatively frequent basis, simply going unreported. Furthermore, these assaults have grown increasingly more complex and effective over the past few years. With an increased understanding of how systems work, intruders have become skilled at determining weaknesses in systems and exploiting them [4]. As new security measures are implemented, new attacks begin to appear.

Various research studies have been previously conducted as a means of characterizing and identifying attacks and assault tools in hopes of developing strong countermeasures. Through forensic analysis and the compilation of large attack databases, security engineers, analysts, law enforcement officials, and others have hoped to design processes capable of determining attack types as well as detecting attacks based on intrusion detection system signatures. Research conducted by Ivan Krsul [6], Peter Mell [5], Tom Richardson [7], and others have all focused on compiling vulnerabilities into databases and analyzing the entries for common characteristics in aim of building attack classification taxonomies. For example Krsul describes his software vulnerability analysis as "a framework for the development of taxonomies according to generally accepted principles [that] can be used to develop unambiguous classifications" [6]. By characterizing existing threats and increasing our understanding of the nature of software vulnerabilities, software developers can hopefully improve the design of products to withstand such attacks in the future.

However, unlike those previous research projects aimed primarily at developing only attack and vulnerability taxonomies, the purpose of this project was not only to study how denial of service tools have changed over time but also develop a model that relates different attacks to one another as well as suggesting future attack strategies. By examining DoS attack history, genealogy, and post-mortem evidence together, researchers gain the ability to not only identify existing attacks and possible countermeasures but possibly even predict future attacks in some cases as well. Although attacks have grown increasingly complex over time, many of the same basic ideas and methods for performing the denial of service remain unchanged or only slightly modified. While previous research models had focused on denial of service attacks as singular data points, modeling attacks as growing genealogical trees formed from several different software attacks yields valuable information on recurring themes in DoS attacks. Furthermore, attack tree hierarchies allow researchers the ability to study how software vulnerability exploits have changed, migrated, and increased in complexity over time. By closely analyzing the evidence left after a denial of service assault and correlating it with current and past attack strategies, we gain the ability to make modest predictions on likely new attack scenarios as well as the preliminary countermeasures that can taken to prevent them.

ATTACK TREES

Typically attack graphs serve as a formal model for identifying and analyzing all vulnerabilities for a particular host or network. Constructing attack graphs is a crucial part of doing vulnerability analysis of a network of hosts [8]. Because each path and node in the attack tree represents a slightly different means of achieving the desired denial of service, attack graphs assess the overall availability vulnerabilities of a single network host. More important to this study though, attack trees provide a formal, methodical way of describing the security of systems, based on varying attacks [9]. In other words attack graphs model the means of achieving the end goal of a successful attack through slightly different means of approach. More formally defined,

> An attack graph or AG is a tuple $G = (S, \tau, S_0, S_s)$ where S is a set of states, $\tau \subseteq S \times S$ is a transition relation, $S_0 \subseteq S$ is a set of initial states, and $S_s \subseteq S$ is a set of success states. Intuitively S_s denotes the set of states where the intruder has achieved his goals. Unless otherwise stated we assume that the transition relation τ is total. We define an execution fragment as a finite sequence of states $s_0 s_1 \ldots s_n$ such that $(s_i, s_{i+1}) \in \tau$ for all $0 \leq i < n$. An execution fragment with $s_0 \in S_0$ is an execution, and an execution whose final state is in S_s is an attack, i.e., the execution corresponds to a sequence of atomic attacks leading to the intruder's goal state [8].

In other words attack graphs are data structures used to model all possible avenues of attacking a network. The graph offers a formal methodology for understanding the software vulnerability and how attacks exploit those vulnerabilities to achieve their goals.

More specific to this study however, attack trees are defined as the set of states that identify all possible means of using a particular vulnerability to achieve a denial of service. Attack trees are structures comprised of singular nodes, or exploits, that all have the desired goal of causing a loss of availability. Unlike the general attack graph definition, attack trees do not outline every possible means of achieving the end success state. Instead, the trees illustrate the evolution of a single attack over time, representing the single process the adversaries use to reach the goal state. Unlike their attack graph counterparts, attack trees only represent the genealogy of an attack, or how the attack has shown modifications and grown in complexity overtime.

Thus each attack tree begins with a root node of the goal state, a complete or partial denial of service, and one or more leaf nodes as means of achieving that desired goal. However each leaf node typically parented by another node which typically represents a more classical or conventional approach or a different means of conducting the same attack. For example, Schneier demonstrates the basic principle using the scenario of a criminal wishing to open a locked safe [9]. While there may be several means of opening the safe through destruction, cutting, or explosives, the criminal wishes to open the locked safe through a single means of attack by cracking the combination. A wide variety of methods exist though for learning the combination of the safe as illustrated in Figure 1. It is important to note that several possible methods exist for conducting a single type of attack. The criminal can either steal the combination through eavesdropping or social engineering, or the attacker may possibly find the written combination by some other method. However, every child node stems from the parent attack method of cracking the combination.

Historically many of the means of cracking the safe can change as well. For example, the criminal may gain several new automated lock-picking kits as new technology is developed or changes are made to the safe design. As a result the tree may grow several new child nodes that stem either from previous child nodes or the parent node itself as demonstrated in Figure 2. Such historical and genealogical changes to the attack tree are extremely important to this research as they may demonstrate how vulnerabilities and attacks have changed over time with new software and hardware technology. Even in the modified attack tree where new means of conducting the same attack have been developed, each node becomes a subgoal, and children of that node are ways to achieve that subgoal [9]. More simply, attack trees simply illustrate every possible procedure for using a specific means to launch an attack.

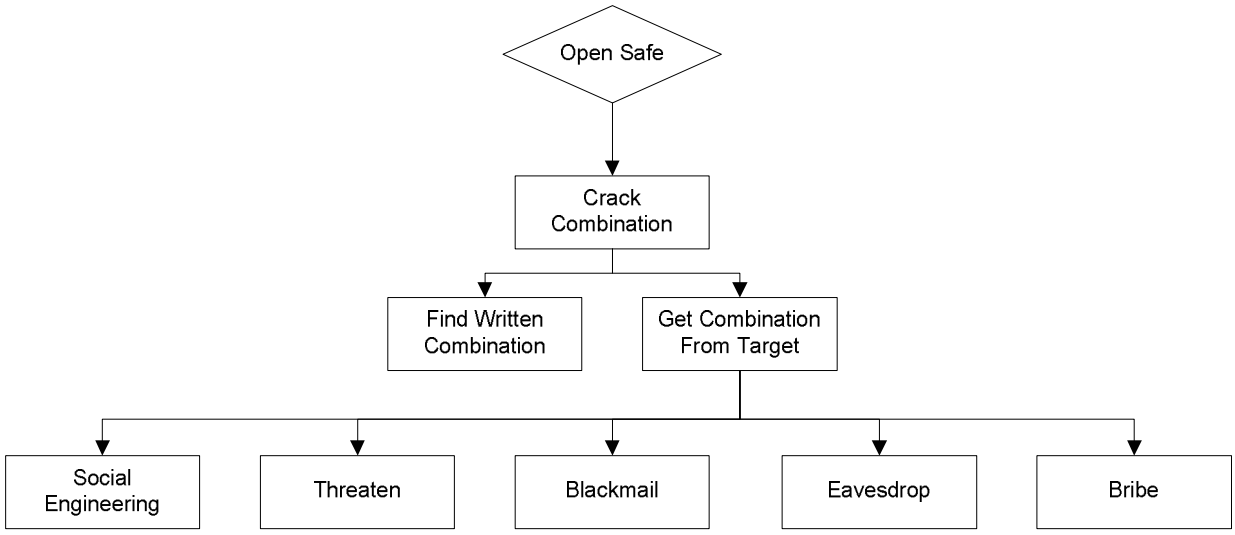

Figure 1. Safe Lock Attack Tree [9]

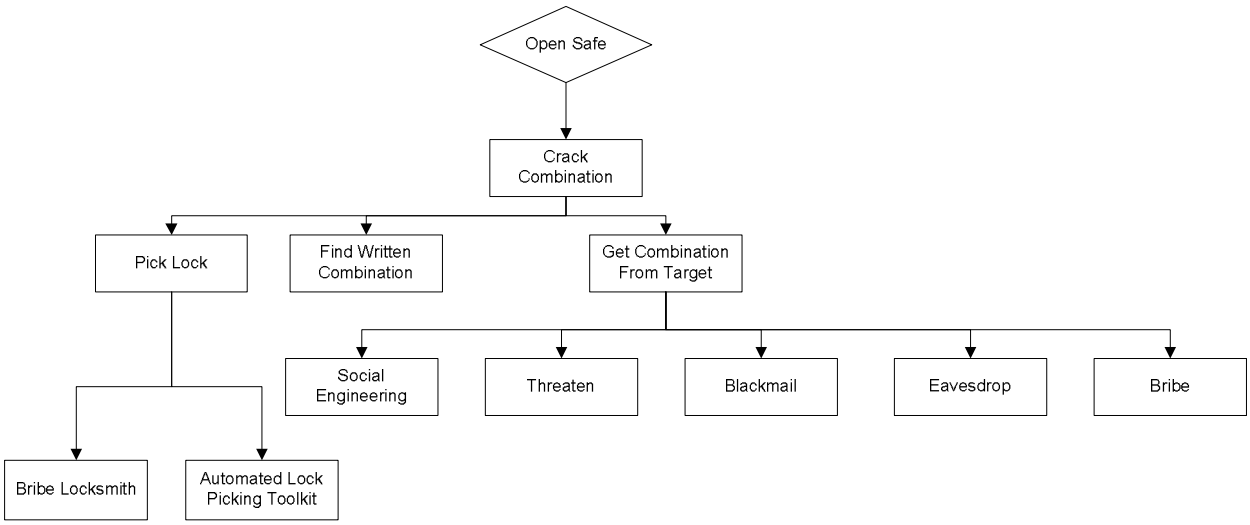

Figure 2. Modified Safe Lock Attack Tree

Unfortunately, developing a genealogical hierarchy of attack evolution is no simple task. Building the attack trees requires close examination of both the vulnerability description, exploit source code, and in many cases forensic evidence. However by building attack trees and studying their growth, evolution, and structure, researchers can determine not only likely methods of attack penetration but possibly predict future modifications to attacks as well. Much like security, attack trees are a process, or methodical method, of describing the vulnerabilities of any system, accounting for those which are most likely to occur, and repeating the entire process in a cycle fashion.

ATTACK ANALYSIS

Previous research conducted by Richardson defines a useful denial of service attack taxonomy for characterizing assault tools [7]. In order to demonstrate the importance of attack trees and their ability to accurately represent threats, we apply the model to each category within the taxonomy. By analyzing attack trees, we gain the ability to study how vulnerabilities have changed over time as well as determine recurring themes in denial of service exploitation.

Specification Weakness Attack Analysis

Specification weakness attacks are those in which the enemy attempts to take direct advantage of a flaw, weakness, or security vulnerability in the design or specification of a network, service, or communication protocol. Most often these exploits actually attack the communication protocol, or more specifically the TCP/IP protocol. Attack tools such as the classic SYN-Flood and LAND attacks specifically target weaknesses in the protocol specification. Because the vulnerability remains at the protocol specification level, the attacks are not only difficult to prevent since doing so requires modifications of an accepted and highly implanted communication standard, but also demonstrate many of the genealogical patterns of improvement and modification important to this study.

As mentioned previously, hosts implementing the TCP/IP protocol stack are limited by the number of half-open connections allowed on any single port at some given time. Unfortunately the TCP/IP protocol requires machine resources such as computing time and memory to be devoted to each of the half-open connections. SYN-Flood attacks work by sending a targeted system a series of connection requests known as SYN packets. Although each packet causes the targeted system to issue a SYN-ACK response in acknowledgement to the connection request, the attacking system never responds to complete the three-way TCP connection handshake. While the victim waits for the ACK completion that follows the SYN-ACK response packet, the system allocates resources to the open port connection request. The resources continued to be allocated specifically to the incomplete connection process until either the attacker responds to complete the connection or and internal timer expires on the original attacker request. Since the attacker never responds to fully complete the three way TCP handshake, resources on the victim are quickly diminished. Once the limit of half-open connection requests has been reached or resources no longer become available, the system will ignore all incoming SYN requests, making the system unavailable for legitimate users.

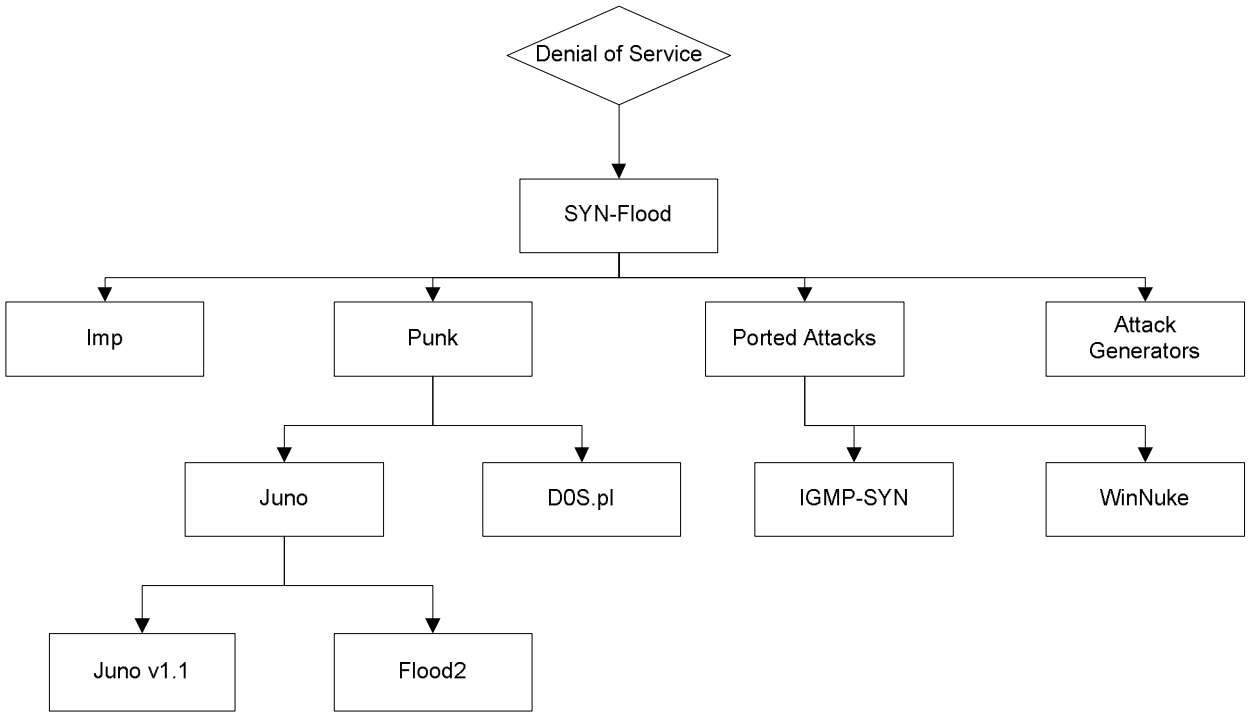

Figure 3. SYN-Flood Attack Tree Genealogy

Since TCP/IP does not incorporate strong authentication in the communication of network hosts, the attack is extremely successful in performing a remote denial of service. Furthermore, there is a requirement for an inappropriately burdensome allocation of memory and computation resources on the target side [10]. Hence for an enemy attempting to perform a successful denial of service on a system, the SYN-Flood attack approach is perhaps an extremely effective and easy means of penetration.

The attack has exhibited significant improvements and variations since its debut several years ago. Originally, forensic analysis of the SYN-Flood attack script demonstrated a simple attempt to open as many incomplete connections on the victim as possible. The attack was effective but inefficient. Soon after examining post-mortem attack data, security professionals easily defeated the original exploit by blocking multiple incoming connection requests from a single host. As the attack tree in Figure 3 demonstrates, the attack quickly evolved over the next few years into several more sophisticated attack tools that still exist as modern means of causing losses in network and resource availability.

Shortly following the introduction of the attack, the author released the more commonly used modern release, known as SYN-Flood 97, while others made the same simple modifications of the original attack code and released their own modified versions including Punk and Slice. The new modifications allowed for source address spoofing that make the attacks even more difficult to filter since incoming connection requests all have different forged sender addresses.

In the next year, several more modifications were made to the Punk attack algorithm including porting the source code to Perl as well as revising the IP spoofing algorithm to be random instead of sequential. Once again many security experts responded by attempting to analyze forged packets and filter similar bogus connection requests. However the more significant changes to the attack came with a revision of the Punk attack script known as Juno. The new TCP SYN-Flooding tool modified the original algorithm for faster packet creation and allowed the attacker to emulate different attacker OS platforms. In other words, the new attack tools allowed for the possibility of more rapid attacks as well as fooling filtering software into determining forged packets had been sent by separate hosts on different operating systems. Once again a short time later, Juno v1.01 was developed that allowed for even faster packet creation, better OS simulation, and improved random source IP generation [11]. In addition a separate work known as Flood2 was also introduced similar to the original Juno that

conserved network bandwidth on the part of the attacker by decreasing the overall packet sizes.

The modifications and evolutionary growth of the SYN-Flood attack tree demonstrates the ability of attackers to improve designs to not only overcome additional security measures implemented by administrators but also improve the overall efficiency of attacks. Furthermore, the attack tree helps to model how similar attacks may appear in other software systems. As attack trees grow, the leaf nodes often spring entirely new methods of conducting a similar attack on either different software systems or communication networks. For example shortly after the SYN-Flood attack was released, several new attacks were developed for the Microsoft Windows platform that simply represented variations of the original SYN-Flood idea that exploits the Out-of-Band data assumption [11]. In the case of the KillWin and WinNuke attacks, the enemy sender directs a special packet to the host with the urgent pointer set. The Windows system responds by immediately allocating resources to the request. However, even though the urgent pointer is set and the Windows system expects data to follow, the attacker instead continues to follow with more Out-of-Band data connections forcing more resources to be exhausted. In fact when sent to the NETBIOS port, the service failed to even handle the requests correctly resulting in an immediate system crash.

Unfortunately, researchers may never be able to fully overcome the problems associated with specification weakness attacks since many are already inherently part of existing standards and intertwined with countless legacy network products. However studying the genealogy of attack trees such as SYN-Flood yields valuable data on how attacks have changed in the past to overcome new security mechanisms and the underlying vulnerabilities which remain to either still be exposed or exploited in new ways. Since the publication of the SYN-Flood vulnerability multiple similar strategies, including LAND and other TCP-Loopback attacks, have stemmed from the same attack methodology. By analyzing themes recurrent in different attack trees we gain some understanding on both the mindset of attacker and the methodology of achieving denial of service.

Brute Force Attack Analysis

Brute force attacks are those in which the enemy attempts to overflow or deplete computing system resources by some forceful and exhaustive means that relies on their own processing power and network bandwidth and the limitations of the system and network resources of the victim machine. Often brute force attacks simply work by flooding the victim network or system with useless data. Attacks such as Smurf, TFN2K, and Stacheldracht all act as extremely effective means of overwhelming the victim systems with vast amounts of maliciously crafted packets.

While some brute force attacks are simplistic and simply overwhelm the victim with ping requests, others demonstrate more sophistication and often will take advantage of implementation or specification weakness vulnerabilities as well. For example, the Smurf attack targets a feature in the IP specification known as directed or subnet broadcasting. By definition the IP protocol specification allows for both unicast and multicast communication transmissions, meaning data packets can be either sent to a single host or group of hosts respectively. In order to send a packet to multiple machines, a machine need only to set the receiver address to the broadcast address. When a packet is sent to that IP broadcast address from a machine outside of the local network, it is broadcast to all machines on the target network (as long as routers are configured to pass along that traffic) [12]. Hence any machine is capable of sending broadcast messages to all machines on a particular local area network.

In order for the enemy to launch a Smurf attack against the network, the attacker simply forges ICMP echo requests using the spoofed sender addresser or the victim. However, unlike a normal ICMP echo request the receiver address is set to a broadcast address. If the routing device delivering traffic to those broadcast addresses performs the IP broadcast function, most hosts on that IP network will take the ICMP echo request and reply to it with an echo reply each, multiplying the traffic by the number of hosts responding [13]. In other words, a ping request is sent to every host on the network with the reply address as a single victim. Once the requests are received potentially dozens of machines then begin sending replies to the target system overwhelming both the network with useless data as well as the victim's protocol stack.

Upon its release, the Smurf attack led to a denial of service of many IRC servers and their providers. Although effective, the Smurf attack demonstrated many of the same inefficiencies as other previous attacks such as the SYN-Flood. Original attacks were blocked simply by preventing IP-directed broadcasts at the router or by disabling ICMP replies on intermediary machines. However, little could be done to prevent the attack on behalf of the victim and as a result the

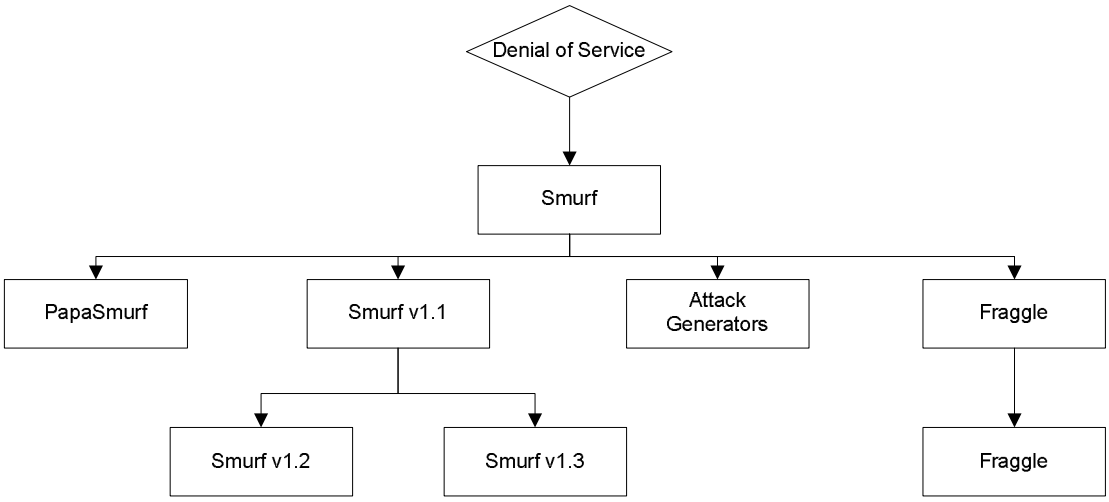

Figure 4. Smurf Attack Tree Genealogy

brute force attack quickly grew in sophistication as demonstrated in Figure 4.

New revisions were released months later that demonstrated faster packet creation by interacting directly with the kernel instead of normal library routines. PapaSmurf and Rurf both modified the original attacks to grow in complexity and efficiency. However the most serious improvements came with newly revised versions of the original source code including Smurf v1.1 and v1.2. Not only did the attack increase in speed but also built in the new capabilities as well. New attack engines were capable of scanning for vulnerable networks as well as vulnerable intermediaries capable of launching the attack against the victim. In fact, version 1.2 even built in sophisticated logging capabilities that enabled the attacker to build a comprehensive resource of vulnerable targets to help launch the denial of service. Finally, version 1.3 ported the attack to the Windows NT platform increasing the ease in which the denial of service attack could be launched by average users. No longer did attackers need access to a UNIX or Linux based platform to cause a loss of availability, nor did they require a sophisticated knowledge on how to compile and launch the attack. In just over a year, the attack had become a user-friendly tool capable of launching an extremely sophisticated attack.

While these analytical results may not appear to be entirely surprising, the research nonetheless demonstrates the ability of the attacker to improve the denial of service tools and adapt to changing security practices, such as filtering IP broadcast traffic at the router. What is more interesting though is the fact that once again the attack was ported to a different platform soon after release. Just as SYN-Flood had sparked a WinNuke attack creation, the Smurf attack was ported to the UDP protocol in a new attack called Fraggle. Much like its counterpart, the Smurf attack cousin, Fraggle, uses UDP echo packets in the same fashion as the ICMP echo packets; it was a simple re-write of Smurf [13]. Likewise, the Fraggle attack also exhibited the same revisions to improve efficiency and effectiveness soon after its release.

Much like specification weakness attacks, there is no means of overcoming brute force attacks. By their very design, computers are intended to process data and output results. By flooding the data input with useless information and overwhelming the system resources, enemies are able to launch quick and effective denial of service attacks. Although researchers may never be able to fully overcome the problems associated with brute force attacks, studying the genealogy of new attack trees may help to predict future attack patterns such as Fraggle. Analyzing the themes in attack trees and trying to predict new or missing branches plays an important role in both understanding security vulnerabilities and exploit efforts.

Implementation Weakness Attack Analysis

An implementation weakness attack is any exploitation of a hardware or software design flaw that directly threatens the availability of system resources. In other words, the attack is an exploitation of security vulnerabilities in the target system. Unlike the brute force and specification weakness attacks, implementation weakness attacks are often easy to repair since the vulnerability exists only on a certain hardware or software package. Once the design implementation is corrected in a manner that patches the

vulnerability, the systems are often able to withstand the attacks. However, it is important to note that this attack category also represents the largest portion of catalogued denial of service tools. Furthermore, as the attack trees will demonstrate, the attack class also shows several recurring themes used by black hats to cause a loss of availability on the target systems.

For example, in late 1997, a major problem was discovered in the way many vendors had implemented the TCP/IP protocol stack. In order to accommodate networks with different maximum data transfer unit sizes and other limitations, the TCP protocol allowed for data packets to be fragmented, or broken into smaller pieces such that each packet contain a small portion of the original data unit. When fragmented packets finally arrive to the receiving host, they are re-assembled and combined to build the original data set. However, the reassembly process implemented by several vendors expected that incoming fragments of a datagram aligned neatly in a way that data at the start of one fragment immediately follows the end of the data set in the preceding fragment. Of course, normally fragmented packets do arrive neatly and reassemble as expected, but the Teardrop attack deliberately crafted packet fragments to cause reassembly problems.

Normally as new fragmented packets arrived, TCP implementations would calculate the amount of memory to allocate to the new packet by taking the difference between the end pointer of the newly arrived packet and the offset of the previous packet according to the Figure 5. Hence under normal reassembly procedures, the packets align neatly without spaces or overlaps. However, the Teardrop attack works by sending the victim specially crafted packets such that the calculated value for the new end pointer is actually less than the previous offset pointer. This can be achieved by ensuring that the second fragment specifies

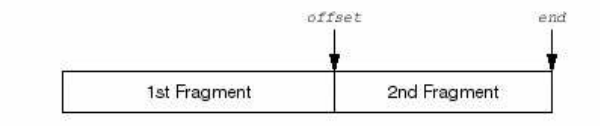

Figure 5. TCP Fragment Re-Assembly [14]

a fragment offset that resides within the data portion of the first fragment and has a length such that the end of the data carried by the second fragment is short enough to fit within the length specified by the first fragment [14]. Although the TCP specification does not allow for this problem to occur, the data packets are not properly validated by the TCP stack implementations on the victim machines.

As a result, the implementation module performing the memory allocation for the newly arrived packet attempts to copy the data into a buffer already assigned to the previous packet. Furthermore, the new total calculated length of the combined data packets actually returns a negative size value. Since the implementations expect an unsigned integer, the negative size value is actually interpreted as a very large positive integer value [14]. Upon such requests, most TCP implementations would either fail due to stack corruption or cause a complete system halt and in both cases allowing the attacker to achieve the goal state of denial of service.

Again much like the SYN-Flood and Smurf attacks, the Teardrop attack originally appeared as an attack script to launch from Linux and UNIX hosts. Unfortunately, unlike other denial of service attacks, the Teardrop could not be easily filtered by firewalls and other Internet gateways since fragments are only reassembled by the intended end receiver. Protection against the attack was reliant upon the victims to patch the appropriate software and firmware vulnerabilities on the affected machines. In the meantime as original software patches were issued by vendors to correct the problem, the Teardrop attack grew rapidly in sophistication as shown in Figure 6.

Originally the Teardrop attack had been coded as a proof-of-concept exploit. In other words, the original tool was a simple script to demonstrate the effectiveness of the attack as a denial of service as well as a means of testing vulnerable systems. The attack was quickly modified though to produce a spoofed source address to help hide enemy identification. Eventually the attack also implemented smaller padding sizes for the first packet as well in order to reduce the overall payload size and increase the overall attack efficiency. In addition, tools such as Teardrop2 implemented faked UDP data lengths as a means of beginning to attack multiple protocol stacks on the victim machine as well as porting the same attack to a new platform.

Perhaps more interesting though is the Bonk branch of the Teardrop attack tree. Soon after vendors issued patches to repair the original Teardrop vulnerabilities, the black hats responded by varying the original attack slightly. Instead of setting the fragment offset to point to a memory segment already occupied by another data packet, the fragment offset is set to point to a location far beyond the end of the data packet. The Bonk attack causes the target machine to reassemble a packet that is too large to be reassembled thereby causing a fault in the network stack [15]. As vendors scrambled to once again prevent the newly discovered attack, many administrators began to

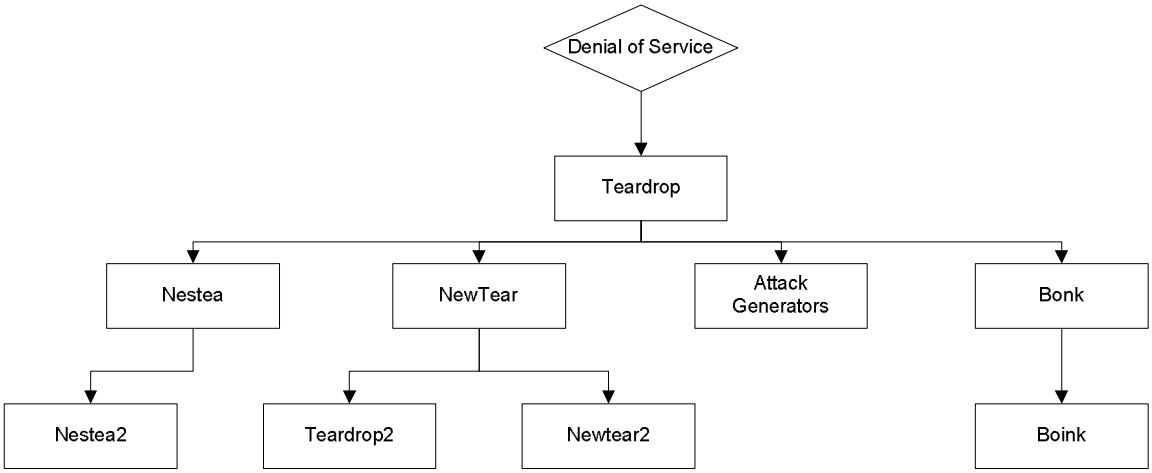

Figure 6. Teardrop Attack Tree Analysis

prevent the attacks by preventing access to the specific port the Bonk tool attempted to exploit. Shortly after, Boink was released to allow attackers to specify port ranges, scan vulnerable hosts, and improve the speed at which malicious packets could be generated.

To make matters worse, several other IP fragmentation denial of service exploits exhibit the same themes in the attack trees. For example the Ping-of-Death attack originally published in 1996 continued to be used in several other forms, most notably Jolt and SSPing, over the course of the next few years while vendors continued to make the same mistakes in design of the ICMP fragment reassembly implementations and attackers continued to use the same methods of improving attack efficiency and effectiveness. Studying the genealogy of this tree and comparing it to other similar attacks yields valuable information on recurring patterns in software vulnerabilities as well as exploitation scenarios.

ATTACK TREE TRENDS

Although several key attack trees have been discussed individually, several more key important themes portrayed in the graphs have not been discussed. In addition to growing in sophistication and complexity, many identified attacks demonstrate other genealogical growth as well. For example most of the attacks discussed thus far were published between 1996 and 1998. However, in 1998 important new attack ideas began to surface that heavily borrowed from successful denial of service exploits to date. Both attack generators and distributed tools demonstrated the continued growth of attacks in complexity as well as hierarchically.

Growth in Sophistication

As nearly every attack taxonomy demonstrates, exploits grow in sophistication over time. In the most simplistic model attack trees, an implementation weakness is discovered and an attack is published which effectively exploits the vulnerability to cause a denial of service. After the attack is published, other malicious code authors borrow from the original idea and, more often, the source code to make improvements. These improvements typically include building a friendlier attack interface, making the attack easier to use, increasing the attack effectiveness, or in some case modifying the attack to continue to function on patched systems.

For example, the Trash denial of service exploit was originally released as a simple tool capable of generating spoofed ICMP packets with random error codes and types. Because the ICMP specification had not been correctly implemented on the Windows operating system, the spoofed ICMP packets often caused a complete system failure. A few months later after the vendor had taken the appropriate steps to repair the vulnerability, Trash2 was released which incorporated new improvements in attack efficiency as well as demonstrate the ability to now generate IGMP packets as well. Both improvements may have arguably been predicted by security researchers though the study of similar attack trees and their growth.

Likewise, exploit tools are often soon ported to either different target or attacker platforms after their initial release. For example, the implementation weakness taxonomy

demonstrated a popular attack known as Teardrop. Over the course of nearly a year, the Teardrop attack spawned several other important exploits. However while the original teardrop attack focused primarily on attacking MS Windows systems, a newly developed attack known as Nestea used the same idea to attack Linux, PalmOS, HP JetDirect Cards, and several other popular platforms.

In any case, many attack trees demonstrate a growth in sophistication, complexity, as well as versatility in very short time periods. Given the common themes demonstrated specifically in the implementation weakness trees, vendors, software programmers, and engineers should not only be analyzing their own products and computing systems for undiscovered vulnerabilities to improve overall security but may also benefit from analyzing the attacks currently being used against other products as well. By studying how a specific attack is being used against one platform, researchers may indeed be able to test the attack on other systems as well and more importantly predict possible changes in the attack that may overcome the newly issued system patches.

Attack Generators

In every attack category, exploits grew rapidly in sophistication. Unfortunately, as several important implementation and specification weakness attacks were being published throughout 1996 to 1998, many malicious coders began to build automated attack tools that simply borrowed from each of the previous effective attack ideas. The new tools, often called attack generators or aggressors, once again showed a growth in complexity and sophistication but little in the way of efficiency.

New attack generator tools exhibited two important themes: combination of old attacks and randomization of new exploitation efforts. Early attack generators were often simply a conglomeration of older and previously published attacks. In other words the next logical step for attackers was to combine multiple denial of service exploits into one tool using simple Unix shell scripts [16]. For example the popular Targa attack generator relied on eight already released attacks as shown in the attack tree in Figure 9. In this case the Targa attack was simply a new interface to several effective attacks, making it multi-platform and extremely effective at causing resource loss on various target systems.

Since many attackers did not completely understand the vulnerabilities that made the attacks possible nor the specific platforms the vulnerability existed on, the Targa attack probably appeared as an even more alluring tool for launching a denial of service. A tool like this has the advantage of allowing an attacker to give a single IP address and have multiple attacks be launched increasing the probability of successful attack [16]. While this particular feature of Targa and other attack generators arguably made the new denial of service tools easier to use and more likely to be abused by those without a strong working knowledge of security, the combination of attacks did result in losses of efficiency on behalf of the attacker. Using tools such as Targa, the enemy now had to launch several completely different types of attacks against an opposing system of which few, if any, the target may have been vulnerable to.

To make matters worse for the attacker, many of the new generators began to incorporate every version of a single attack. Instead of simply including the newest revision of the SYN-Flood or Teardrop attack, even the malicious coders developing the attack generators demonstrate little knowledge on the how each individual script actually functions and what vulnerabilities it exploits. For example, DataPool was released as several different versions, the latest of which includes both Nestea and Nestea2 as well as Bonk and Boink. Hence the combination of old exploits in attack generators often leads to severe efficiency penalties as the same attacks are continuously repeated in several different fashions.

On the other hand, many attack generators exhibit an attempt on behalf of the source code author to possibly uncover new vulnerabilities and attacks by randomizing the denial of service effort. For example, another attack generator known as Aggressor as well as the forth and final release of Targa, both generate useless TCP/IP packets with random invalid fragmentation sizes, window sizes, lengths, types, and other unusual characteristics in attempt to discover unreported specification or implementation weakness attacks. Fortunately in most cases, these attack generators alone remain far from being efficient or effective.

Attack generators demonstrate both the ease of use of new exploit tools as well as their reduction in overall efficiency. Nonetheless, the models demonstrate the importance once again of recognizing vulnerabilities that have already been discovered on other platforms and ensuring the proper safeguards are implemented for any system under one's own direction. As attacks are combined and randomly launched against unsuspecting hosts, the victims become susceptible to a wide variety of possibly unpublished vulnerabilities on the target platform.

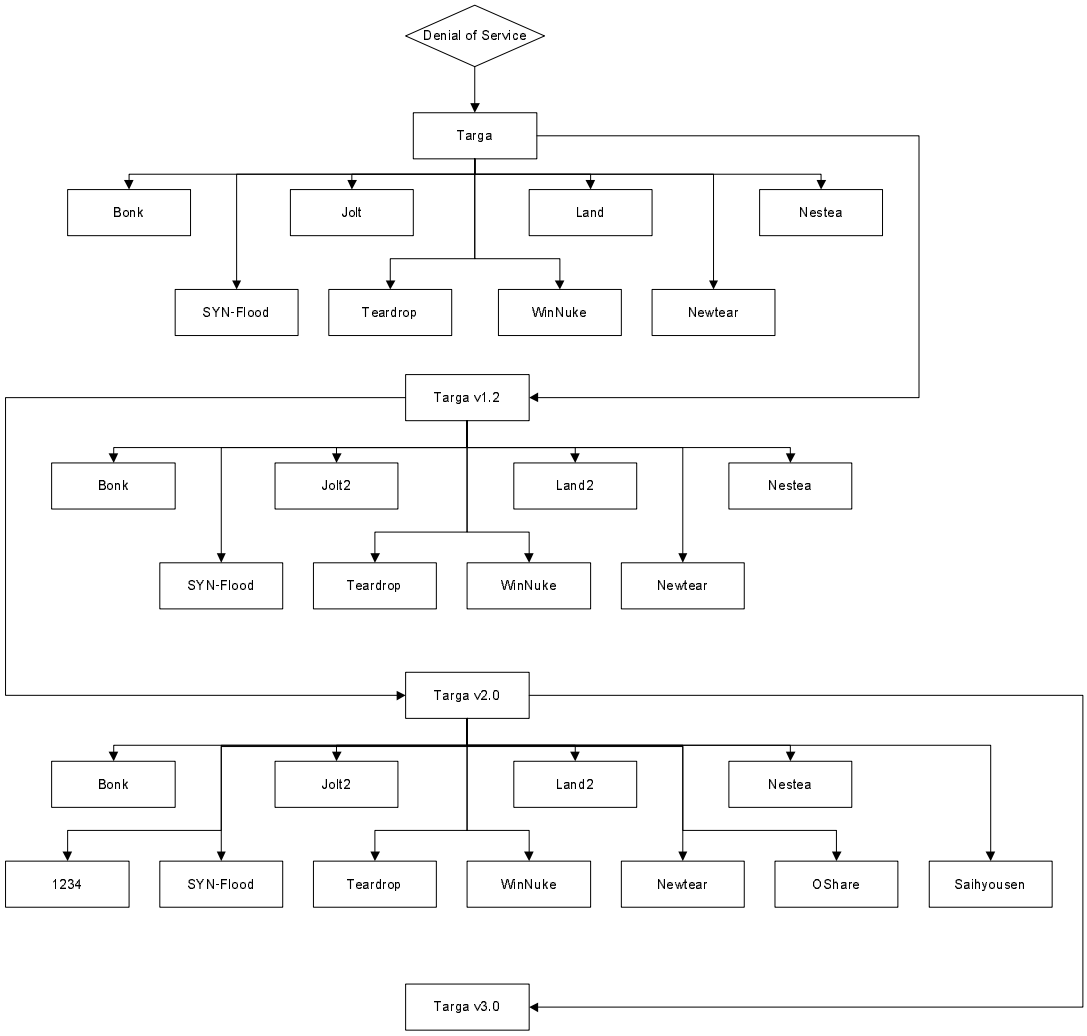

Figure 9. Targa Attack Tree

Distributed Attacks

Perhaps the most disturbing trend in the growth of attack trees is the combination, automation, and distribution of denial of service tools. Although it was not entirely possible to prevent specification weakness and brute force attacks, most Internet hosts increased in network bandwidth, processing power, as well as memory resources to a point where it became difficult for a single attacker to overwhelm the target system with enough data that it would be unable to respond to other legitimate requests. As previous attack methods began to meet their limits in less than two years, the next logical step was taken to combine the power of a number of multiple systems into a distributed denial of service cluster [16]. In 1999, several new attack tools were published that combined the strengths of both specification weakness and brute force exploits in a new coordinated and distributed denial of service interface.

By installing denial of service attack clients on multiple compromised systems and developing a hierarchy of control such that all clients could be issued commands from a central location, attackers could now launch their attack from multiple hosts. The first two tools to appear once again borrowed heavily from previously published attack tools. For example, Trinoo simply coordinated a simplistic UDP flood. By coordinating an attack by several dozen client hosts each generating literally hundreds of UDP datagrams, the attacker could now overcome the limitations of previous attacks by brute force means.

Hence many of the attack trees previously discussed spawned new nodes that demonstrated both the automation and distribution of new tools.

Previous attack tools were quickly modified to incorporate attack distribution. For example, Targa was modified to create Tribe Flood Network, or TFN. Much like its Targa cousin, TFN supports ICMP flood, UDP flood, SYN-Flood, and Smurf style attacks [17]. In addition other attacks quickly demonstrated growth in sophistication once again incorporating encryption and other stealth techniques to evade capture and removal of clients.

More importantly, the attack trees now exhibited nodes that were arguably part of both the brute force and specification weakness category according to the original taxonomy. Distributed denial of service attacks signify a major change in overcoming the previous resource limitations of the attacker. Since the power of many is greater than the power of few, coordinated and simultaneous malicious actions by some participants can always be detrimental to others if the resources of the attacker are greater than the resources of the victim [18]. New distributed attacks simply brute forced many of the previously successful attack strategies. Since intelligence and resources were no longer located on the same network, both bandwidth and computing resources were conserved on the part of the attacker.

Although the distributed attack tree branch may have been somewhat hard to predict given earlier models, many of the future branches of the distributed denial of service attack nodes may not be as necessarily as hard to predict. For example, soon after the release of Trinoo and TFN, Stacheldracht and TFN2K appeared incorporating new methods of encrypting client and attacker communications to ensure that information remain hidden and to prevent normal session hijacking. In addition both attacks modified the original attack algorithms once again for a slight improvement in efficiency. Just as previous attack trees demonstrated incorporation of other attacks, scanning tools, and other features, the new distributed attack tools may soon begin to exhibit the same characteristics as well.

COUNTERMEASURES AND FUTURE WORK

While several measures currently exist as means of helping to mitigate the effects of denial of service attacks including load balancing, resource throttling, attack detection, containment, and filtering, there currently exists no full solution for entirely protecting a system against the threat of loss of availability. For example, CERT and several security vendors all advise several appropriate measures a vulnerable host can take in order to help prepare and withstand such an attack. However, given the specification weaknesses in widely used communication protocols such as TCP/IP and the inherent vulnerabilities of any host to a brute force attack, no formidable solution may exist in the near future that will entire protect systems from denial of service attacks.

In computer and network security, very few solutions actually exist though that entirely prevent many types of attacks. Instead security is an ongoing process of risk assessment. Improving the security of a system requires the users to determine where the current vulnerabilities exist, which are most likely to be exploited, and which represent the greatest risk. Fortunately, attack trees provide a formal methodology for analyzing the security of systems and subsystems [9]. In other words, attack trees form a groundwork for understanding the process of improving security.

Typically attack graphs are used by security professionals in order to determine every possible means of penetrating a vulnerable system. By trying to determine every method of obtaining the attack goal, security engineers and network administrators can then determine which nodes in the tree have the least cost to the attacker in terms of both time and dollars, require the least skill on behalf of the attacker, and are the most easily accomplished. By sending appropriate values to each node, administrators can often determine likely paths of system penetration and the possible attack scenarios for an enemy to successfully reach the final goal.

Much like their attack graph counterparts, genealogical attack trees demonstrate possible scenarios for achieving some type of attack such as a denial of service. More importantly though, attack trees represent the path that malicious users have already taken in order to successfully achieve a goal. The genealogical hierarchy of an attack tree portrays not only how attacks have changed over time but also how they have *not* changed. As demonstrated by the Teardrop attack tree, several important paths of attacking vulnerable implementations of the TCP/IP protocol stack existed. However, at the time the initial software patches were released only one particular method of attack was currently being used by the enemy. Soon afterwards as the attack started to become ineffective against many repaired target machines, a new branch of the attack tree spawned which exploited the same fragmentation vulnerability as before by simply another means.

Studying the genealogy of the fragmentation exploits and other implementation weakness attack trees, software designers and forensic analysts can potentially determine what other undiscovered vulnerabilities may exist. For example, had software designers fully analyzed the current attack tree structure of the Teardrop attack

and the corresponding post-mortem attack data they may indeed have noticed other branches within the attack tree that were currently missing as part of the hierarchy. In other words, existing attack trees can be analyzed in conjunction with forensic data by software engineers as a methodical approach of attempting to determine other future methods of attack, likelihood of exploitation, and possible new attack trends. By analyzing the current structure and genealogy of the tree and attempting to determine currently missing branches, designers gain a valuable tool in helping to prevent future vulnerabilities. Just as administrators and security engineers use attack graphs as a methodical approach to representing important risks currently exhibited by a computing system, attack trees can be used by forensic analysts and patch developers to help determine new attack trends and software vulnerabilities.

Software vulnerabilities exist because it is impossible for designers to produce systems which conform to extremely strict security standards without formal proofs and validation of the source code. Although many designers take careful steps to prevent software vulnerabilities, vendors simply can not predict what flaws will later be discovered. Thus attack trees became an extremely important mechanism for building more secure software. Should designers analyze the current attack against the system using a more methodical approach and determine other likely methods of attack as well as existing attack improvements, they may indeed uncover other vulnerabilities that would otherwise not be noted until later.

In addition, attack trees provide an effective means for other vendors to determine whether their products are currently vulnerable to attacks being launched on other platforms. Several of the attack trees demonstrated that attackers later ported the attack to a wide variety of other platforms each of which was also susceptible to risk. Attack trees provide a way to think about security, to capture and reuse expertise about security, and to respond to changes in security [9]. Given many of the common themes demonstrated in the implementation weakness trees, engineers may be able to gain valuable information on undiscovered vulnerabilities in their own software designs simply by analyzing the attacks currently being used against other products. By studying how a specific attack is being leveraged against one particular software application, researchers may indeed be able to test the attack on other systems as well. More importantly researchers may gain the ability to predict possible changes in the attack that could possibly overcome the newly issued system patches.

CONCLUSION

As the connectivity of modern computers continues to increase and Internet infrastructure continues to expand, the importance of maintaining the availability of network and computing resources has become an extremely important and challenging task. Recent distributed denial of service attacks demonstrate the serious vulnerabilities that still remain within much of the world's electronic communication system and the crippling effects of simple to sophisticated attack agents. Although no absolute countermeasure against the attacks currently exists, analyzing the patterns, themes, and genealogical growth of current attacks yields valuable information on existing software vulnerabilities and possible future attack trends.

By examining denial of service attack trees coupled with forensic evidence of the attacks, researchers gain the ability to not only identify existing attacks and possible partial countermeasures but possibly even predict future attacks in some cases as well. Although attacks have grown increasingly complex over time, many of the same basic ideas and methods for performing the attack remain unchanged or only slightly modified. Many of the same attack themes that have been uncovered in the past are reappearing in new forms which target different software platforms, improve attack efficiency, or distribute the attack among multiple hosts. By studying the genealogy of existing attack tools and their hierarchical structure coupled with post-mortem analysis, researchers are afforded the ability to study how software vulnerability exploits have changed and migrated over time.

BIBLIOGRAPHY

[1] Moore, D., Voelker, G.M., and Savage, S. (2001) *Inferring Internet Denial-of-Service Activity*. Proceedings of the 10th USENIX Security Symposium. USENIX.

[2] Corcoran, E. (1996). *Hackers strike at N.Y. Internet Access Company*. The Washington Post, Sep. 12, 1996.

[3] CERT Coordination Center. (1997). *Denial of Service Attacks*. Pittsburgh, PA, Carnegie Mellon Software Engineering Institute.

[4] Kumar, S. (1995). *Classification and Detection of Computer Intrusions*, Ph.D. Thesis. Purdue University.

[5] Mell, Peter. (1999). *Understanding the Global Attack Toolkit Using a Database of Dependent Classifiers.* Proceedings on the 2nd Annual Workshop on Research with Security Vulnerability Databases.

[6] Krsul, I. (1998). *Software Vulnerability Analysis*, Ph.D. Thesis. Purdue University.

[7] Richardson, T.W. (2001). *The Development of a Database Taxonomy of Vulnerabilites to Support the Study of Denial of Service Attacks*, Ph.D. Thesis. Iowa State University.

[8] Sheyner, O. Haines, J., Somesh, J., Lippman, R., and Wing, J.M. (2002). *Automated Generation and Analysis of Attack Graphs*. Defense Advanced Research Projects. [On-line]. Available: http://www-2.cs.cmu.edu/afs/cs/project/calder/papers/sp02/paper.ps. (Date Accessed: 19 July 2002).

[9] Schneier, B. (1999). *Attack Trees: Modeling Security Threats*. Dr. Dobb's Journal. [On-line]. Available: http://www.ddj.com/documents/s=896/ddj9912a/9912a.htm. (Date Accessed: 19 July 2002).

[10] Schuba, C., Krsul, I., Kuhn, M., Spafford, G., Sundaram, A., and Zamboni, D. (1997). *Analysis of a Denial of Service Attack on TCP*. Proceedings of the 1997 IEEE Symposium on Security and Privacy.

[11] Packetstorm Security. (2001). *Denial of Service Attack Tools.* [On-line]. Available: http://www.packetstormsecurity.com. (Date Accessed: 19 July 2002).

[12] CERT Coordination Center. (1998). *Smurf IP Denial of Service Attacks.* Pittsburgh, PA, Carnegie Mellon Software Engineering Institute.

[13] Huegen, C. (1998). Smurf Attack. [On-Line]. Available: http://www.quadrunner.com/~chuegen/smurf.txt. (Date Accessed: 19 July 2002).

[14] Hoggan, David. (1994). *The Internet Book*. Hoggan Website. [On-line]. Available: http://www.theinternetbook.net/. (Date Accessed: 19 July 2002).

[15] WWDSI. (2001). Saint Tutorial: Boink Attack. [On-line]. Available: http://www.wwdsi.com/demo/saint_tutorials/boink.html. (Date Accessed: 19 July 2002).

[16] Dittrich, D. (1999). *The DoS Project's "Trinoo" Distributed Denial of Service Attack Tool*. Seattle, WA, University of Washington.

[17] Dittrich, D. (1999). *The "Tribe Flood Network" Distributed Denial of Service Attack Tool*. Seattle, WA, University of Washington.

[18] Mirkovic, J., Martin, J., and Reiher, P. (2001). A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms. Los Angeles, CA, University of California Computer Science Department.