;login:

THE MAGAZINE OF USENIX & SAGE

December 2000 • volume 25 • number 8

inside:

SECURITY:

A NOTE ON SECURITY DISCLOSURES

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

a note on security disclosures

by Brian Martin

Brian Martin is a Senior Security Engineer with the DSIC Network Security-Group. His team provides penetration assessment and audit for the commercial and government sectors.

<bmartin@attrition.org>

In recent months, a handful of outspoken security professionals have begun to openly challenge the philosophy of full disclosure. For years, most of us in the security community have held this philosophy as a basic tenet of vulnerability management. Software vendors have a notoriously bad track record in handling bugs and vulnerabilities in their products. Rather than practice due diligence and handle the incident swiftly and openly, vendors have not given these matters the attention they deserve. Open communication between security consultants or hackers who find these vulnerabilities and the vendors they report to is more of a myth than anything, despite what the vendors would like you to believe. Because of this, full disclosure has kicked in, causing security types to release all of the details to public forums. In some cases, they include working exploit code as proof of concept, to "gently" prod the vendor into dealing with the vulnerability in short order.

There are essentially three levels of disclosure with regard to security-vulnerability information seen today:

1. General information indicating a vulnerability in a specific software package or operating system. Often the information is extremely vague and not adequate for administrators to fix the problem themselves. Advisories from CERT and similar outfits fit in this category.
2. Full technical information, often in the form of an advisory. Technical details enable security personnel to understand the problem fully and often fix it without any additional information. No exploit code or "proof-of-concept" code is included. Security firms and consultants typically release in this fashion.
3. Working exploit code. This is sometimes accompanied by a technical advisory explaining the vulnerability in depth. Other times it may only include a few notes in comments at the top of the program.

The recent argument suggests that including exploit code while disclosing vulnerabilities has a downside that severely outweighs any upside. The perceived upside is that by including working exploit code, vendors are forced to respond to the issue quickly, lest thousands of their customers remain vulnerable to the new exploit. While this indeed puts extra pressure on the vendors to fix the problem, it potentially leaves thousands of systems exploitable afterward. While software patches are available, there is nothing forcing administrators to install them to negate the problem. At this point, full disclosure becomes a two-edged sword.

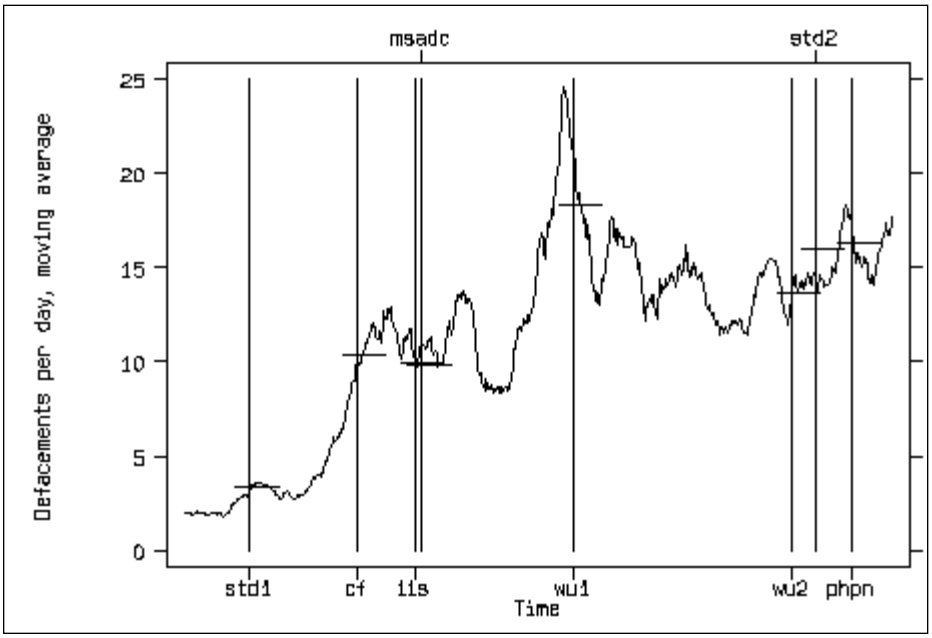
The downside to all of this is that these exploit scripts and utilities are available to anyone with a shred of computer know-how, who now has the capability of breaking into foreign systems. This in turn leads to the "script kiddie" phenomenon: large numbers of mostly unskilled wannabe hackers use the tools to break into (and often deface the Web pages of) systems around the Internet.

Much to the joy of some of these outspoken security professionals, there exists hard data to back their claims. For months, they have been making these claims with no real backing to support their arguments. As with any person making claims with no proof, it is often easier to discount their words as a bit fanatical, especially when their tone is elitist and condescending. Using data collected by Attrition (*<http://attrition.org>*) along with BugTraq archives at Security Focus (*<http://securityfocus.com>*), several

cases have clearly emerged that demonstrate the cause and effect between exploit-code release and Web-page defacement.

Before examining this data, several things should be considered. This is not the silver bullet killing any and all doubt you may have had in the outspokens. Rather, this is data that proves one piece of the puzzle. As with all things, this is not a black-and-white issue. Other issues must be considered in addition to the ideas presented above.

Several cases stand out in the past two years that demonstrate the downside to releasing exploit code. Listed below are eight vulnerabilities that allow an intruder to gain some form of elevated privileges remotely. Included with each is the approximate date exploit code was made public, along with other thoughts or comments.



Vulnerabilities, November 1, 1998 - September 30, 2000
Note: Horizontal lines are the average defacements per day 14 days before and 28 days after vulnerability became "public" knowledge.

"std1" – automountd/statd remote buffer overflow (UNIX) Jan 4, 1999 *<http://www.securityfocus.com/archive/1/11788>*. Shortly after public disclosure, there was a small spike in defacements per day. In the following months, an incredible growth began.

"cf" – Cold Fusion l0pht advisory w/exploit code (NT) Apr 20, 1999 *<http://www.securityfocus.com/archive/1/13377>*. Other problems in the Cold Fusion package came to light in Phrack 54 (Dec 25, 1998), but did not include detailed exploit information. Based on the graph, it seems the release of the CF exploit resulted in more defacements per day.

"iis" – IIS Hack eEye advisory (NT) Jun 16, 1999 *<http://www.eeye.com/html/Advisories/AD19990608-3.html>* and *<http://www.securityfocus.com/archive/1/15448>*.

"msadc" – RDS/MSADC RFP advisory (NT) Jun 23, 1999 *<http://www.wiretrip.net/rfp/p/doc.asp?id=1&iface=2>*. The combination of IIS Hack and the MSADC exploit being released at approximately the same time led to two small spikes. Because of difficulty in getting the exploit code to work early on, it is believed that the incredible spike in the following months was more indicative of the exploits being public. During this time, a large percentage of defacements mirrored by Attrition appeared to be NT-based and mostly a result of the MSADC vulnerability.

"wu1" - wuftpd 2.5 remote buffer overflow (UNIX) Nov 20, 1999 *<http://www.securityfocus.com/archive/1/35828>*. While the average number of defacements per day dropped steadily shortly before and after its release, there was another noticeable spike shortly afterward. Once again, it is believed that the delay was caused by initial problems in using the first versions of the exploit code. In the weeks after its release, more versions of the exploit came out, increasing the chances of successful exploitation on a remote host.

"wu2" – wuftpd 2.6* remote buffer overflow (UNIX) Jun 23, 2000 *<http://www.securityfocus.com/archive/1/66367>*. As seen before, a small increase can be

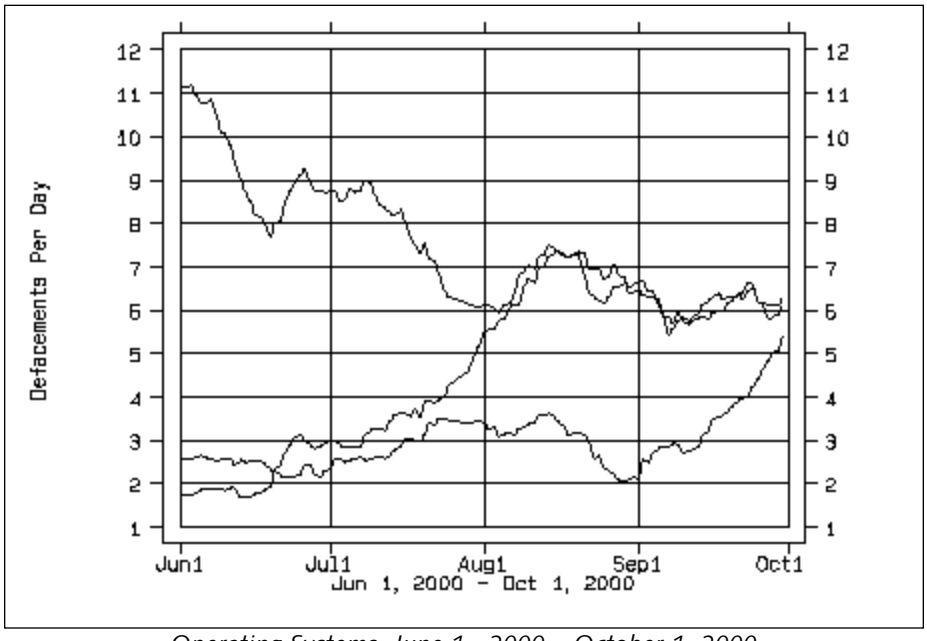
seen before and after the release of the exploit code. Running into the approximate release of "std2," the upward growth became even more noticeable.

"std2" – statd remote buffer overflow (UNIX) Jul 16, 2000 *<http://www.securityfocus.com/archive/1/70306>.*

"phpn" - PHP-Nuke news site administration Aug 21, 2000 *<http://packetstorm.securify.com/0008-exploits/PHP-Nuke.c>.* Once again, a noticeable spike shortly after disclosure of the exploit information. During this time, a large percentage of defacements reported to Attrition were a result of this exploit. Because the attackers could post custom messages to a news application and not replace the entire page's content, it was rather easy to identify which defacements were a direct result of this vulnerability.

While these eight examples are fairly clear, it should be noted that with the disclosure of any remote exploit code, defacements tend to increase shortly afterward. Depending on the operating systems affected, ease of use of the exploit, and availability of vulnerable machines, the numbers do not always shift so dramatically. Working with the Attrition mirror on a daily basis makes one more aware of this trend.

A key part of the definition of "script kiddie" is the lack of technical skill s/he possesses. It is widely believed that most script kiddies use Windows machines and favor exploitation of Windows NT servers. In the cases where a UNIX-based exploit is simple enough to use (easy to compile, simple command-line arguments, etc.), script kiddies will shift from exploiting Windows machines and begin to attack UNIX systems. The best example of this can be seen in the recent "wu2" (wuftpd 2.6) and "std2" (statd) vulnerabilities. Not only did significant spikes occur shortly after public disclosure of exploit code, but a radical shift in the overall number of Windows and UNIX operating systems being defaced occurred.



Operating Systems, June 1, 2000 – October 1, 2000
(Top Line – Windows NT, Middle Line – Linux, Bottom Line – All other OSs)

Despite a steady increase in Windows NT defacements for the last year, NT systems were defaced less often shortly after the two UNIX exploits were released. In keeping with this, Linux (exploit code for wu2/std2 was written to exploit Linux primarily) defacements climbed dramatically. Defacement groups that had previously been dominant on Windows platforms suddenly began to deface more and more Linux machines.

While the data points to a conclusion that publicizing exploit scripts is harmful to the Internet community, that may not necessarily be the case. Because Web-page defacement is a public event and a strong motivation of many script kiddies, it provides a method to extract data to show the trends and statistics above. However, one must consider the sequence of events and results of exploits being created but *not* being posted to a public forum.

No fewer than 20 defacements occurred in the process of writing this article.

Unpublished exploit scripts are like currency in the hacker subculture. The power of a single remote exploit that is unknown to vendors enables a single person to potentially break into thousands of machines, often with no recognizable trace of how it was done. Many intrusion-detection systems will not recognize the fingerprints of these new exploits. The hackers who hold these scripts typically do not deface Web pages or perform any action that would draw undue attention to themselves. To do so would bring more attention to their actions and create a better chance that someone would figure out how they are compromising machines and what the new vulnerability is.

In some cases, these exploits circulate in the underground for up to a year before being made public. If even a dozen hackers have such an exploit while actively using it over a one year period, the damage becomes negligible compared to a few dozen Web defacements that occur as a result of an exploit being made public. While it is unfortunate to see a site hacked as a result of the public disclosure of a vulnerability, it really becomes one of the necessary evils in doing so. One has to balance the sites being hacked in one hand versus a vendor patch that will allow thousands of sites to be protected against the exploit.

As I wrote this article, I had constant reminders of everything covered above. No fewer than 20 defacements occurred in the process of writing this. No doubt some occurred as a result of exploit scripts being made public, fueling the script kiddies. Equally so, some of these probably occurred from classical attacks such as password guessing, sniffers, or nonpublic exploits that haven't crossed BugTraq yet. Is releasing exploit code as evil as some are now saying? I think that is answered by which side of the sword you'd rather be cut with.

[Editor's Note: My home systems were penetrated within two hours of the Kerberos exploit being published. I think it was the first BSDI had heard of it. Even having the fix within four hours wasn't enough to prevent the exploit. RK]