

# Efficient visualization of change events in enterprise networks\*

Andrew Stewart

June, 2006

## Abstract

*Change* is a crucial property from a security perspective.

The detection of change underpins many of the operational security activities that organizations typically carry out. For example, the essence of security monitoring is to detect changes, then analyze those changes in the context of the applicable security policy.

Security tools are available to perform change detection at a host level, such as Tripwire (Kim, 1994). Such tools typically employ a local software agent, and identify changes that occur in the filesystem of the host.

We describe a tool that performs a similar role in a network environment. The tool employs a variety of visualization techniques to efficiently communicate changes that occur in enterprise networks.

## 1 Introduction

Every change that occurs within an environment has the possibility that it could affect the security of that environment. Generally, we expect detrimental effects. Muffet (1995) describes how the *effective* security of a host will diminish over time. Factors that contribute to this “decay” in security include:

- Changes made by system and network administrators that are intended to be temporary, but that are subsequently forgotten. (“I’ll just bring up FTP for a minute to allow me to

transfer these files...”)

- Configuration errors, the effect of which may not be immediately apparent. A router will continue to pass traffic if an access control list intended to block traffic is not applied to the necessary interface, for example.
- Unintended consequences caused by complexity arising from the interaction between different elements within the environment. This could be software products with overlapping functionality, or personnel with overlapping roles. Difficulty in understanding the interaction between multiple configuration settings can also lead to errors.

The above are examples of situations where change can occur unintentionally. Change can also occur legitimately within an organization, through proper channels such as the change control process. But even if such business processes are robust, the problem of accidental change remains, as does the problem of *rogue* and *malicious* change. Examples include:

- Users who violate security policies, such as by attaching a wireless access point to the network.
- System administrators or application developers who introduce systems or network services into the network, without passing them through standard boarding procedures.
- Worms, viruses, bots, or trojans that infect systems for some purpose (typically malicious).

---

\*Presented at the IEEE Workshop on Enterprise Network Security, Baltimore MD, August 28, 2006.

- Insiders who plant backdoor or trojan software on systems.
- Denial of service attacks that render systems or services unavailable, either as an end-goal or as a tactical portion of a larger attack.

In enterprise environments, the challenge of detecting and responding to such change events can be exacerbated by factors such as geography, language barriers, and the strict separation of business functions for legal reasons.

How to proactively exercise control over such an environment is an unsolved problem. One approach is to spend some portion of security efforts on constructing detection “nets” that can identify behavior that has bypassed a policy “gate”. The ability to monitor change on a network is an example of one such conceptual net.

The rest of the paper proceeds as follows. In §2, a set of functional requirements for network change detection are proposed. A tool designed to meet those requirements is presented in §3, along with the various approaches to visualization it employs. The implementation of the tool is described in §4. §5 discusses the operation of the tool in the presence of a hostile adversary, and conclusions are drawn in §6.

## 2 Requirements

At a high level, we seek to answer the question: *what has changed on the network?* We define three overarching functional goals:

1. *Ability to detect network change*

We define a ‘network change’ as three possible events: an addition to the population of the network, the addition of a service offered by a system on the network, or the unavailability of a system that was previously available. These are events that have the possibility of changing the security posture of the network by introducing vulnerability, or that might indicate the result of a denial of service attack.

2. *Agent-less collection of data*

To install a software agent on every host, subnet, or VLAN within an enterprise environment is a non-trivial task. (The information security team within the organization may not even have the jurisdiction to do so.)

Additionally, if a rogue host is attached to the network in an unauthorized act, then the host in question is unlikely to incorporate the standard security policy verification agent.

For these reasons, agent-less methods of information gathering are the preferred method.

3. *“At-a-glance” communication of change*

A significant number of change events will occur over time within any enterprise network. The reporting method must therefore be efficient. This immediately suggests the use of visualization and “small multiple” techniques to represent the data in a form that allows efficient exploration and understanding.

To-date, academic research into the use of visualization for computer security has tended to focus on the analysis of network traffic (such as in McPherson, 2004 and Lakkaraju, 2004) or the output of monitoring systems such as IDS (such as in Girardin, 1998 and Oline, 2005).

Due to the requirements described, our focus in this work has been on the visualization of change events that occur and that are visible *directly on the end-points within the network*, and not on the traffic that flows between those end-points.

## 3 Design

Netdiff<sup>1</sup> is a tool designed to meet the requirements described in the previous section. In this section, the user experience is described. The details of the underlying implementation are described in §4.

To communicate network change, netdiff employs several custom methods of visualization. The primary method of visualization uses simple ASCII text. The question might reasonably be asked: *why use text-based graphics in this modern age?* The answer is to promote efficiency.

ASCII text can be delivered within a simple HTML page, and also within email and terminal environments. This “plain text” approach to visualization is therefore more flexible than one that is tied to a particular language, and subsequently a particular set of platforms. The rendering of ASCII text by a web browser, terminal, or email client is also less computationally expensive than a visualization

---

<sup>1</sup>The name “netdiff” is an allusion to the `diff(1)` command on Unix-like platforms.

based on OpenGL or Java. This speaks to the overarching goal of efficiency.

The method of visualization represents IP address space as a series of ASCII characters. The period (“.”) character represents one IP address. Characters that neighbor each other horizontally are contiguous in IP address space terms, and are displayed in the mode of written english language (interpreted left to right, top to bottom). For example, the visualization below could represent the 24 IP addresses in the range from 192.168.1.1 to 192.168.1.24.

```
.....
.....
.....
```

For each IP address, netdiff maintains a FILO queue of information. To detect and communicate change, netdiff compares past and present entries in those queues. The following table maps possible state changes to their symbolic representation within the visualization.

past	present	symbol
down	down	.
up	up	o
down	up	+
up	down	-

“up” and “down” correspond to whether the device at the IP address could be contacted on a particular occasion. “down” therefore refers to both a device being unavailable (such as when a host is taken offline for maintenance), and also to the case where an IP address is unassigned.

The visualization below conveys information on changes that have occurred within a block of 24 IP addresses.

```
.....+.
oo.....
.....-...
```

Using the symbol key, it can be seen that there are currently three live devices within this address range. Two devices remained in a constant state. One device that used to be present could not be contacted, and one new device appeared. (The determination that a device “remained in a constant state” is made only on the basis of the properties of

a device that netdiff understands and can measure, as described in §4.1)

From a security perspective, the appearance of a device on the network has a greater security implication than one’s absence. A device that has been decommissioned or that is currently unavailable does not have a security implication per se (other than when security requirements are viewed as a pure subset of reliability requirements). The *addition* of a device such as a wireless access point or an unpatched server could certainly have a security implication.

In compliance with the definition of “network change” noted in §2, netdiff also provides the ability to detect when devices have changed in some way. The various properties of a device that netdiff understands are described in §4.1, but includes listening TCP ports. If a device opens a new TCP port, netdiff displays that device within the visualization using the “x” character. This is useful for monitoring situations where the desire is to maintain a highly static environment in terms of the services that hosts offer, such as in a DMZ.

### 3.1 Scalability

An important measure of quality for a technique of visualization is not only how effectively it can convey the underlying data, but whether that communication remains effective as the number of data scale upwards.

Offloading “processing” to the human brain is an approach that all techniques of visualizations employ, fundamentally. By doing so, the hope is to enable patterns and relevant information to be more easily recognized by the user. In the visualization below, network change information on an IP addresses space containing approximately 500 addresses is shown.

```
.....o.....ooo.....o.....
.....
.....ooxoooooxoo.....oo.....oo.....
.....ooo.....
.....o.....
.....+.....xx.....
.....xooo.....oo
ooo-.o.....o.....
.....oo.....o.....-.....
.....o.....ooo.....+.....
.....
```

Visual clutter has been reduced by muting secondary elements, and color has been used to highlight change events. We have successfully used this form of visualization to identify and display network change events on an IP network with several thousand live nodes.

When such a visualization is rendered in HTML, the amount of information displayed can be controlled with a combination of changing the font size in the web browser, and sizing the browser window itself. Most browsers have keyboard shortcuts that make the text on the page bigger or smaller, and this makes it straightforward for the user to change the “resolution” of the visualization. The web browser has in many respects become the ubiquitous client, and this approach is intended to leverage that expectation of familiarity. Another advantage is that web browsers typically dedicate a large percentage of their window real estate to the display of the web page itself, and not to administrative debris.

The horizontal stratification displayed in the visualization is a by-product of the left-to-right interpretation of address space employed. As an observation, IP address usage on corporate networks is typically more sparse than might be expected.

Within the HTML rendering created by netdiff, individual devices can be clicked to display their change history. This allows the user (such as a security analyst) to examine the details of what change occurred, and then to make a value judgment as to how to proceed. An example of this output for a particular IP address is shown in Fig 1.

### 3.2 Outlier analysis

In an enterprise network, the task of attempting to identify those changes that are rogue, accidental, or malicious is complicated by the sheer number of legitimate changes. Ideally, that signal to noise ratio could be reduced so that the relatively small number of events with a serious security implication could be identified.

One approach is to set an expectation and then flag exceptions. Netdiff can be preset with knowledge of the properties of the standard platform builds in use within an environment. Expectations can also be set for certain IP address ranges.

If the requirement for web servers within a par-

ticular DMZ is to only serve web content over SSL, then netdiff will not flag the addition of a server using TCP port 443, but will flag the use of TCP port 80. Using this method, netdiff can detect the undesirable *decay* that tends to afflict systems over time, as they drift from their desired build configuration.

Another example of this approach is in detecting outliers within the list of OS types in use. If the corporate standard within the enterprise is Microsoft Windows, then the appearance of Linux box is of particular interest (and vice-versa in the opposite scenario). Consider the netdiff output below, regarding a particular LAN segment.

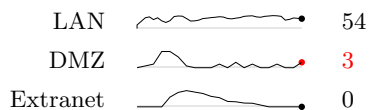
```
40 Microsoft Windows 2003 Server
33 Microsoft Windows 2000 Pro
7 Cisco 3600 router running IOS 12.2
4 Cisco Localdirector load balancer
2 Linux 2.4.18 - 2.6.7
2 Cisco AS5200 terminal server
1 Linksys WET-11 wireless Ethernet
```

The purpose of this deliberately minimalist report is to attempt to increase the probability that the operator will notice an anomaly – such as with the wireless access point in the example.

Traditional network vulnerability assessment tools attempt to report as many properties of the remote systems that they scan as possible; the intention being to report any condition that might have a security implication.<sup>2</sup> In contrast, the philosophy employed in the design of netdiff is to draw attention to change events, where those changes may be statistically small.

### 3.3 Placing network change in historical context

It is useful to have contextual information available when attempting to identify anomalies. Netdiff can display trends in the quantity and rate of change, as shown here:



<sup>2</sup>This approach has been driven in large part by market forces, as vendors of network vulnerability assessment tools compete on the information gathering capabilities of their products.

status	timestamp	mac	os guess	tcp ports
up	29/06/2005@22:01	unknown	Linux 2.4.0 - 2.5.20	53,80,443,32770
down	28/06/2005@20:54	n/a	n/a	n/a
up	27/06/2005@21:35	unknown	Linux 2.4.0 - 2.5.20	53,80,443

Fig 1. Display within netdiff of the reverse chronology of change associated with a particular IP address.

In the example, a LAN, DMZ, and extranet are being monitored. The number of change events that occurred during the last time period are displayed on the right of the visualization.

The number of change events on the LAN reflects DHCP allocation of addressing. The extranet can be seen to have undergone a significant change event, followed by a long tail of follow-on activity.

Netdiff uses the same historical data to perform alerting based on calculated thresholds. These thresholds are useful for detecting wide-scale change events, such as with a worm infection.

The *rate* of change can itself be incorporated within the ASCII-based visualization that netdiff employs. For example:

```

...o....
.o..X...
..X...o..

```

The size of each character signifies the degree of change in comparison to prior activity. This historical context can help to draw the operator's attention to more unusual changes, rather than for all changes to have the same weight within the visualization.

## 4 Implementation

Netdiff comprises approximately 3000 lines of Perl. Invocation is from the command line, in one of two modes: data collection or reporting.

In data collection mode, netdiff employs a number of network scanning techniques, and populates a database with the results. Reporting mode uses the contents of the database to display results.

### 4.1 Database format

The data that netdiff collects are stored in an ASCII flat file database. A hierarchical storage model layered on top of a filesystem was considered (such as

the one described in Muffet, 1995), but a flat file was chosen for simplicity.

Each line in the database file corresponds to data on one IP address. Internally, netdiff builds a hash table, indexed using these IP addresses. Every line in the database begins with an IP address, and is followed by one or more records.

A record stores data that reflects the state of a network device associated with a particular IP address at a particular moment in time. Square brackets are used to delimit a record, and internal fields are separated with the “|” character, as shown here:

```
[1|31/06/2005@20:01|x|Cisco Soho 97
router running IOS 12.3(8)|23,80]
```

Within a record, the first field stores whether the device responded to a ping request (1 for yes; 0 for no). A date and time stamp for when the data in this record were collected is stored in the second field.

The third field stores the MAC address of the device, if available. Typically, a MAC address can only be obtained when the host used to scan the remote device is no greater than one hop away. If any field lacks data (as with the MAC address field in the example record shown above), an “x” is substituted.

The fourth field stores a guess of the remote operating system. The OS of a device can sometimes be remotely inferred through an understanding of subtle differences in responses to particular types of TCP/IP traffic across platforms. (See Fyodor, 1998.)

The fifth and final field stores the TCP ports that were responsive when the record was constructed.

### 4.2 The logic of network change detection

Recall from §4.1 that each line in the ASCII database used by netdiff contains data on one IP address. When a new record is added, it is appended to the

existing text on the applicable line in the database. The last record on each line is therefore the freshest from a chronological perspective, and the penultimate record is the next most recent.

To detect change, netdiff compares the two latest records for every IP address that is in scope for the current query. If the most recent record differs from the previous record, then a change has occurred.

Consider the following line from a netdiff database:

```
192.168.1.10[1|30/06/2005@22:01|x|x|22,443]
192.168.1.10[1|31/06/2005@16:20|x|x|22,23,443]
```

There are two database records associated with 192.168.1.10, meaning that data on this IP address have been collected on two occasions. The text can be made easier to read by removing the leading IP address, and stacking the two records vertically:

```
[1|30/06/2005@22:01|x|x|22,443]
[1|31/06/2005@16:20|x|x|22,23,443]
```

Based on the date and time stamps, the records contain data for consecutive days. On the second day, the device at the IP address began listening on TCP port 23, in addition to TCP port 22. For a piece of network infrastructure such as an edge router, the likely desire is for it to be administered using only encrypted protocols (such as SSH). The addition of the Telnet service on this device is therefore a change that would need to be investigated and corrected.

## 5 Evasion strategies

In the design and implementation of any security mechanism, it is valuable to consider the scenarios in which the methodology employed would not be effective.

The common approach to attacking any security system is to understand the assumptions on which the system operates, and then confound those assumptions.

One evasion strategy would be to monitor the regularity with which netdiff performs a sweep of the network. If a cadence is detected, it would be possible to hide a network change by removing it during the regular times when netdiff is used to perform a

network scan. The initial mitigating strategy would be to introduce randomization to the scanning interval.

There are a finite set of properties that netdiff can detect for remote devices. One way to evade netdiff would be to make a change that netdiff is unable to recognize. For example: install a trojan that listened on a UDP port, rather than a TCP port. A similar attack would be to remove the daemon from a listening TCP port, and run a new service on that same port. Without banner collection or service fingerprinting, netdiff could not recognize that a functional change had taken place. That functionality could be built.

In addition to these evasion strategies, the classic “training problem” also exists, in which it is uncertain whether the initial baseline that is gathered includes any attacker activity that was already present.

The attacks described above assume a hostile adversary. Users or system administrators who make changes for convenience-sake and that happen to violate a security policy are unlikely to put such forethought into their actions. Similarly, changes that are accidental are not designed to be stealthy. Performing network change detection still has value therefore, for the detection of those scenarios.

Experienced security practitioners rely on multiple compensating controls to provide protection, detection, and response. Like any control, netdiff cannot be relied upon in isolation.

## 6 Conclusions

Historically, the detection of change as a network property has been the focus of network availability monitoring software, rather than for security purposes.

The focus of efforts to-date in the field of network vulnerability assessment have been on increasing the comprehensiveness of vulnerability detection. (A progression driven in large part by competition within the market for those products.) We have outlined an alternative philosophy, in which the approach is to detect change events, rather than states of known vulnerability.

Traditional efforts in the sphere of security visualization have focused on the visualization of network traffic or security monitoring data. We have described an approach that focuses on the visualiza-

tion of the properties of the end-systems within the network.

When looking at the behaviour of a particular system, there will be certain features that can be noticed by sight, and certain other features that can only be detected by having a program perform a computational analysis. The ASCII-based visualization method we have implemented within netdiff provides the former of these approaches, and the 'outlier analysis method' the latter. In doing so, the goal has been to enable the efficient communication of network change in enterprise environments.

Oline, A., Reiners, D. (2005), *Exploring three-dimensional visualization for intrusion detection*, Proceedings of the IEEE Workshop on Visualization for Computer Security (VizSEC 05), 2005.

## 7 References

- Fyodor. (1997), *The Art of Port Scanning*, Phrack Magazine, Volume 7, Issue 51, 1st Sep, 1997.  
See: <http://www.insecure.org>
- Fyodor. (1998), *Remote OS detection via TCP/IP Stack FingerPrinting*, Phrack Magazine, Volume 8, Issue 54, 25th Dec, 1998. See: <http://www.phrack.org>
- Girardin, L., Brodbeck, D. (1998), *A Visual Approach for Monitoring Logs*, Proceedings of the Twelfth Systems Administration Conference (LISA 98) Boston, Massachusetts, December 6-11, 1998.
- Kim H.G., Spafford E.H. (1994), *The design and implementation of tripwire: a file system integrity checker*, Proceedings of the 2nd ACM Conference on Computer and Communications Security, 1994.
- Lakkaraju, K., Bearavolu, R., Yurcik, W. (2004), *NVisionIP - A Traffic Visualization Tool for Security Analysis of Large & Complex Networks*, Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security, 2004.
- McPhearson, J., Ma, K., Krystosk, P., Bartoletti, T., Christensen, M. (2004), *PortVis: A Tool for Port-Based Detection of Security Events*, Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security, 2004.
- Muffet A. (1995), *WAN-hacking with AutoHack - Auditing security behind the firewall*, Proceedings of the 5th USENIX Unix Security Symposium, 6th June, 1995.