Global Information Assurance Certification Paper

TITLE * MERGEFORMAT
Creating Your Own SIEM and Incident Response Toolkit Using Open Source Tools

GIAC (GCIH) Gold Certification

Author: Jonathan Sweeny, jsweeny@iu.edu
Advisor: Rob VandenBrink

Accepted: 20 June 2011

Abstract

This paper describes how one can use open source tools to create an incident response toolkit. A significant piece of your toolkit is a Security Information and Event Manager (SIEM), or the ability to store and process event logs. Two reasons you may want to create your own toolkit and SIEM are: financial and the ability to customize. In addition to outlining what software tools you should have in your kit and how to create them, I will explain how to prioritize your efforts in creating the toolkit. This paper could easily be used to guide in the selection of a commercial SIEM.

Introducing an Incident Response Toolkit

When an information security analyst is investigating incidents, he or she needs to have an organized set of tools. Similar toolsets are found in other occupations; such as a doctor's surgical tray or a mechanic's hand tools. Though an information security investigator may have a toolbox with items such as write-blockers, media and antistatic bags, this paper will focus on software (rather than hardware) tools. Having the right collection of software tools can help to decrease the time required to identify a system responsible for any questionable network activity and to isolate it from the network. An incident response toolkit can automate repetitive tasks, provide useful information to other IT professionals, and permit them to assist in remediation.

SIEM: The Core of the Toolkit

What is a SIEM?

A Security Information and Event Management (SIEM) solution is the core of an information security worker's incident response toolbox. The SIEM began as a product to collect event logs from various systems into a central server but has grown to also detect and act on certain types of behavior and to track compliance.

Why Create Your Own SIEM?

If you have talented developers, you may be able to save money by developing a SIEM in-house. In some situations you are forced to create your own solution because there is no budget available to purchase a supported product. "There exists today a rich selection of tools that can perform all or some of the features of a full SIEM at little or no cost, other than the time and energy to learn, install, configure, and customize them" (Miller, Harris, Harper, Vandyke & Blask, 2011).

Another reason to create your own SIEM is to customize your solution – either because

your infrastructure is very complex or because your needs are different from most commercial SIEM customers. One example of a unique environment is a university where IT services are often decentralized.

A third reason to create your own SIEM is that you develop programming and tool set skills in your staff. Additionally, when you create something yourself, it is much easier to diagnose and correct problems.

Reasons *Not* to Create Your Own SIEM

If your homegrown SIEM will be vulnerable to SQL injection and cross-site scripting, then you are better off not having a SIEM. You must ensure that developers have secure coding training and experience so that any solutions you implement do not increase your risk. Visit HYPERLINK "https://www.owasp.org" https://www.owasp.org to learn more about web application security and secure coding practices. Their development guide, *A Guide to Building Secure Web Applications and Web Services*, is a fantastic overview of secure coding principles.

If phone support, documentation, and maintenance from a vendor are important, then you should not create your own SIEM. If the service goes offline because of a hardware failure or a software misconfiguration while the developer of your SIEM is out sick (or worse, after he or she has left your company), that would be a bad time to realize that you do not know anything about the architecture or how to bring it back online. If you do create your own SIEM, ensure that more than one person knows how to support and fix bugs, that code is documented, and that a revision control system is in use.

How to Create a Toolkit and SIEM

This section will outline how a SIEM can be created using open source and other free tools, drawing on the experience of Indiana University over the last seven years.

Collect Logs

The first step in creating a toolkit is to begin collecting relevant network connection logs and network flow data. Before you start collecting, consult your legal counsel to determine the appropriate retention schedule for these logs. You may decide to preserve these logs for as short as one week or as long as a year or longer – each business has different needs.

Commonly these log types include DHCP leases, VPN sessions, Active Directory domain controller logs, and various other authentication logs. In addition to logs related to authentication, collect historical data related to network traffic such as firewall logs, NetFlow, and sFlow. These records contain source and destination IP addresses, source and destination ports, IP protocols, TCP flags, number of bytes/packets, and more.

Initially, you will likely use a tool like syslog to collect these logs and store them in text files. Syslog was developed by Eric Allman for the sendmail project, and became the default way Unix and Linux systems send and store logs (Costales & Allman, 2002). In addition to logging locally on a server or workstation, syslog is commonly used to send logs over the network to a log collection server. If Windows systems will be sending or receiving syslogs, then Snare (http://www.intersectalliance.com/projects/index.html) is an option.

At first, you can use command-line tools like grep (or findstr in Windows) to search the log files on the server, but you'll soon want to begin transferring the logs into a database. When using grep consider taking advantage of the "-i" option to ignore upper/lower case (dvader versus DVader) and the "-w" option to search based on "word boundaries" (generally when searching for "192.168.2.3" you do not want to see results for "192.68.2.34", for example).

Storing Logs in a Database

Storing logs in an indexed database will permit easier and quicker searching of the logs. For example, you might have a record in a DHCP table that contains the start time that a Media Access Control (MAC) address was leased an IP address and the time

that the lease expired. It would then be trivial to search for the DHCP leases that any one MAC address had or the many MAC addresses that were leased a certain IP address.
One method to import the logs into the database is to push the logs received via Syslog into the database using a scripting language like Perl. See Appendix A for sample code. Alternatively the service that is the source of the logs may be able to push the logs straight into the database, depending on the services and the type of database. Some database types to consider would be MySQL, Oracle, and Microsoft SQL Server. Your database administrators may have a type of database they prefer, so you may not be given an option. You may want to work with your database administrators to create indices to speed up queries – at the cost of slowing down inserts. The fields you will want to index are the ones that are most commonly queried: probably IP addresses and usernames.
If a scripting language is used to import the logs into a database, they should be normalized at the same time. An unmodified ("best evidence") copy of the logs should be retained in text format in case it is needed for evidence in a trial or hearing. Text normalization can help to make querying of the logs less confusing and more efficient. The first field type to consider normalizing is usernames, because it is easier to ensure that they're all imported lowercase than to remember to always query the field using a lowercase search parameter. Strings can be converted to lowercase in SQL by using the LOWER function or like this in Perl using regular expressions:

```
$string =~ tr/[A-Z]/[a-z]/;
```

The second field type to consider normalizing is timestamps. It is best to store timestamps in the Coordinated Universal Time (UTC) standard to avoid confusion about various time zones and daylight savings time. Be aware that timestamps in logs will often be in a local time zone rather than UTC. See appendix B for sample Perl code that converts timestamps to the UTC standard.
Potential issues with importing logs into a database are size and scalability. Some logs

are enormous and may be resource intensive to insert, index, and query. Be sure that you consider and implement a retention policy for logs stored in databases.

Searching the Logs

Once you have your logs stored efficiently in a database, the next step is to create a stored function that is given an IP address and timestamp and provides details about the system assigned that IP address. This function will search through the database tables and other directory information retrievable via lightweight directory access protocol (LDAP). The function will then return details about the computer and user responsible for the system including: computer name, MAC address, DHCP lease start/end, full name, email address, office location, phone number, department/division, etc.
This sample Perl, DBI, and SQL code could be used to determine the MAC address that had the lease of a particular IP address at the time of an incident. Note the use of bind parameters to mitigate the threat of SQL Injection.

Using your system of network accountability (possibly a NAC or NAP solution, or something like NetReg, HYPERLINK "http://netreg.sourceforge.net/" http://netreg.sourceforge.net/), you will discover the user responsible for that MAC Address, generally by determining a username or a distinct employee ID number. Using that information, you can search directory information (such as LDAP) to determine values such as full name, email address, phone number, and department. The following sample Perl code could grab the department name of a user from a Windows domain controller via secure LDAP:

When you have completed this code, you will be able to take the IP address and timestamp provided by your intrusion detection system or an external report (e.g., SpamCop or DMCA) and easily track down the device and user responsible. Keep in

mind that timestamps in these reports may not always be in the same time zone as your logs. Appendix B provides sample Perl code to convert timestamps to and from the UTC standard.

You may eventually use these stored functions via a web service so you should be sure to sanitize any input received and use bind parameters in your SQL statements. See the OWASP *SQL Injection Cheat Sheet* for more details, including examples for Oracle, MySQL, and SQL Server:

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Quarantine and Blocking Tools

Once the ability to automate identification has been improved, the next step is to make it easier to place network blocks. For example, instead of manually using the Active Directory Users and Computers snap-in of the Microsoft Management Console (MMC) to add usernames to a group of users denied from using the VPN service, create a reusable script that will perform the same function using LDAP. To make the best use of your incident response staff's time, create scripts to automate as many tasks as possible. These scripts can be grouped together to automate identification, notification, and quarantine.

Whenever a network block is placed or a notification is sent, record that action in a database. You'll be able to use this data to track recidivism, identify trends, and provide metrics. Again, consider data retention policies for these records.

Notification

When a device or user needs to be blocked from the network, notification should be sent to the user, the IT staff in that department, or both parties. Instead of composing these notifications manually, prepare canned messages for several notification types. These messages can contain placeholders that your code replaces with details pertinent to the incident at hand. In addition to explaining what activity was detected (the reason for

the block); these messages should provide remediation information and instructions. In some cases the notification may be as simple as “contact the helpdesk”, but in other situations they may provide detailed instructions to backup data, wipe the hard drive and reinstall the operating system.

Most issue tracking systems have functionality built in to support pre-defined frequently used messages. The open source Request Tracker (RT) calls this feature RTFM (RT FAQ Manager). Issue tracking systems will be discussed further in the Incident Metrics section of this document.

Provide Lookup Tools for the Helpdesk

In order to help users quickly remediate system compromises or network blocks, the helpdesk will benefit from having information about the detected event that caused the quarantine of the system. Depending on your environment and the amount of trust you have in your helpdesk, you may provide a minimal amount of information (e.g. the block reason being “FTPd on non-standard port”) or as much as the packet capture and signature details from your intrusion detection system.

Incident Metrics

Incident metrics could be stored in your SIEM or in a separate tracking system. A tracking system (such as Best Practical’s open source Request Tracker: http://bestpractical.com/rt/) can be used to document incidents and related correspondence and metrics. In most issue tracking systems, tickets can be linked to one another and custom fields can be created and used.

In addition to the obvious benefits of tracking correspondence and actions taken, these systems provide the ability to track related events and trends. A repeat offender, for example, may need extra attention or a referral to a disciplinary office.

Verizon Enterprise Risk and Incident Sharing (VERIS) Framework

One option for tracking incident metrics is the Verizon Enterprise Risk and Incident Sharing (VERIS) framework *(Verizon, 2010)*. Verizon developed a set of metrics for the purpose of "capturing key details surrounding a breach" based on these four categories**:**

Agent (whose actions affected the asset)
Action (what actions affected the asset)
Asset (what assets were affected)

Attribute (how the asset was affected)

This framework makes it incredibly easy to collect very useful metrics. For example, you could display a chart that compared external hacking incidents where the confidentiality of a server was compromised against internal misuse incidents where the usability of the network was affected. Whether or not the VERIS framework is used, incidents should be categorized and metrics reviewed at least quarterly. In addition to highlighting current threats and challenges, metrics can improve resource allocation and point out tasks whose automation via scripts would improve efficiency.

Charts

An easy way to visualize metrics is by creating charts. Charts can benefit in many ways. If displayed on a monitor in a Security Operations Center (SOC), charts make it very easy for an information security analyst to notice a spike or a change in a trend, such as a large number of authentication failures. Over a longer period (e.g. months or quarters) charts can be used in a quarterly report to management to demonstrate worth or to warn about new trends.

Three easy to use chart packages are: pChart, Google Image Charts, and Google Chart Tools. pChart uses PHP and is open source. It is a great option if PHP is your favorite programming language. Google Image Charts creates live image files on a Google server in response to the data being provided as parameters in the URL. This is probably the easiest way to start creating charts, but has limitations on image size and URL length. Google Chart Tools, the search giant's newer solution, works with any programming

language and uses JavaScript and HTML to display the data. Its images are a little fancier (mouse-over interactions) than the Google Image Charts and do not have limitations on image size or number of data points.

Determine which metrics are most useful to chart (e.g. number of incidents, incident type or severity). Appendix D contains Perl code that creates Line Charts using Google Chart Tools. Consider using charts in quarterly reports or cycling on a display in the Information Security Operations Center.

Keep Expanding the Toolkit

In addition to the tools mentioned above, there are likely some tools that could really benefit the particular environment you work in. Note the procedures your incident response staff routinely undertake and explore how those tasks could be improved or automated by adding tools or scripts to your virtual toolbox. A few samples are listed here:

Event Correlation and Information Sharing

Event correlation is the process of connecting certain events from a large number of events. A good example of event correlation is the detection of a brute force authentication attack. Once a SIEM is brought up and information about security events is being stored in a database that is easy to query, then it is time to begin performing event correlation. Event correlation can be as simple as keeping track of a list of IP addresses known to be used by miscreants and then alerting when activity is detected from those IP addresses.

The open source Simple Event Correlator (SEC, HYPERLINK "http://simple-evcorr.sourceforge.net/" http://simple-evcorr.sourceforge.net/) is a simple little tool that reads from log files and correlates events based on a configuration file. SEC can write to a file or execute a program in response to events triggering a rule.

Many issue tracking systems come with an API or another method of interacting with it

from another system. RequestTracker, for example, comes with a RESTful web service which your event correlation solution can use to create new tickets.

In addition to recording information about threats and attacks, there are good reasons to share that information with trusted partners. Your information may help them protect their network and their information may help you protect yours. Information Sharing and Analysis Centers (ISAC) like the IT-ISAC and the REN-ISAC exist to promote information sharing about threats. The information sharing is protected by nondisclosure agreement. Information sharing has been so useful that these ISACs are developing infrastructures to aid the information sharing process. The REN-ISAC's Security Event System is one such example: HYPERLINK "http://www.ren-isac.net/ses/" http://www.ren-isac.net/ses/

The FBI's InfraGard program is a system that promotes the sharing of information about threats to the national infrastructure. There is a strong IT security presence in the InfraGard program but it also covers sectors such as agriculture, banking, chemical, energy, and transportation.

Null Route Injection

Using BGP, inject null routes into routers for IP addresses (internal or external) that are misbehaving. This method is preferable to blocking a device at the switch port or firewall because blocks take effect immediately. Network engineers can set up a null route injection API, ideal for security analysts who have the authorization to quarantine devices from the network but do not maintain the network hardware.

Injecting null routes can block a device using DHCP sooner than waiting for the device to be denied a lease renewal. In addition to blocking the MAC address from DHCP renewal, inject a null route for the duration of the current lease.

WHOIS Notification

When an external party is attacking devices on your network, that activity should usually be reported to the service provider responsible for the source IP address. In order to determine who should be alerted, the WHOIS protocol is used. One option is to point your web browser to HYPERLINK "http://whois.arin.net/ui" http://whois.arin.net/ui and manually look up the IP address. The American Registry for Internet Numbers (ARIN), however, will only provide results if the IP address is in the United States or Canada. There are separate Regional Internet Registries (RIRs) for different areas of the world: LACNIC (South America, Mexico, and a few surrounding countries), APNIC (China, India, Japan, Australia and others), AFRINIC (the African continent), and RIPE (Europe, Russia, the Middle East and others).

Instead of using a web browser to manually search each of the registries until you get a result, code can be written to handle this for you. The Perl code in appendix C will return an email address from the WHOIS data when given an IP address.

Password/Passphrase Scramble

In the event that an account is compromised and under the control of a miscreant, having the ability to disable access to an account can be critical. Another option besides disabling the account is to scramble the password or passphrase. One reason the password scramble may be preferable is that email to the disabled account would bounce but would reach an account with a scrambled password just fine. Another reason is that the helpdesk likely has access to set a new password but not to enable accounts. Be sure that you disable any self-service password reset functionality or else the miscreant may continue to abuse the account.

Self-Service Remediation

If your users can be trusted enough, consider creating a web interface where they can view information about blocks or quarantines placed against their devices or accounts. After reviewing remediation information, they can assert they performed the

required steps and initiate a procedure to unblock the device or account. Of course steps should be taken to identify abuse of this trust and to blacklist abusers. Web pages that accept user input need to be particularly thorough in sanitizing input to prevent attacks such as SQL injection and cross-site scripting.

Conclusion

Creating your own SIEM can be a fantastic solution to improving your incident response capability. This is particularly true if you do not have a budget for a commercial option or if a custom SIEM is preferred because of a complex environment.

References

Costales, B, & Allman, E. (2002). *Sendmail*. O'Reilly Media, Inc.

Tipton, Harold F., & Krause, Micki (2009). *Information Security Management Handbook, Volume 3*. CRC Press.

2. Miller, D, Harris, S, Harper, A, Vandyke, S, & Blask, C. (2010). *Security Information and Event Management (SIEM) Implementation.* McGraw-Hill Osborne Media.

3. Verizon. (2010). *Verizon Enterprise Risk and Incident Sharing Metrics Framework*. Retrieved from HYPERLINK "http://www.verizonbusiness.com/resources/whitepapers/wp_verizon-incident-sharing-metrics-framework_en_xg.pdf" http://www.verizonbusiness.com/resources/whitepapers/wp_verizon-incident-sharing-metrics-framework_en_xg.pdf

Appendix A: Importing Logs into a Database

The following Perl function receives DHCP logs (from a dhcpd server) via STDIN and imports them into the database specified.

```
use strict;
use warnings;

use DBI;
use Date::Parse;
use Date::Format;

################################################
## Function for importing DHCP logs into database
sub dhcp {
  ## Connect to DB & create database handle:
  my $dbh = DBI->connect($data_source, $username, $password) or die $DBI::errstr;

  # Compose SQL statement
  my $sql = qq(MERGE INTO $dbname.$dhcplogstable a
              USING (SELECT to_date(?, 'YYYY-MM-DD hh24:mi:ss') datetime,
                            to_date(?, 'YYYY-MM-DD hh24:mi:ss') lease_end,
                            ? ip,
                            ? mac,
                            ? wins
                     FROM dual) inpt
   ON (a.mac_addr = inpt.mac AND a.datetime <= inpt.datetime AND
       a.assigned_ip = inpt.ip AND a.end_time IS NULL)

   WHEN MATCHED THEN UPDATE
   SET a.last_seen = inpt.datetime, a.wins_name = inpt.wins, a.lease_end = inpt.lease_end

   WHEN NOT MATCHED THEN
   INSERT (datetime, lease_end, last_seen, assigned_ip, mac_addr, wins_name)
   VALUES (inpt.datetime, inpt.lease_end, inpt.datetime, inpt.ip, inpt.mac, inpt.wins));

  ## Prepare the query
  my $sth = $dbh->prepare($sql) or warn "prepare failed: $DBI::errstr\n";

  my $_dhcpcounter = 0;

  while(<STDIN>) {
    chomp;
    my $_datetime  = "";
```

```
    my $_lease_end = "";
    my $_ip        = "";
    my $_mac       = "";
    my $_wins      = "";
    warn "\nProcessing Line: $_\n" if $ENV{debug};  # Print if debugging

    ## Grab the values from the log line using this regular expression:
    if (/^(\w{3}\s+\d{1,2}\s+(\d{2}:){2}\d{2}).*DHCPACK on ((\d{1,3}\.){3}\d{1,3}) ends
(\d{4}\/\d\d\/\d\d:\d\d:\d\d:\d\d) to (([A-Fa-f0-9]{2}:){5}[A-Fa-f0-9]{2})/) {
      $_datetime = time2str("%Y-%m-%d %H:%M:%S", str2time($1));
      $_ip = $3;
      $_lease_end = $5;
      $_mac = $6;
    } else {
      next; ## We only want rows that match the regular expression
    }

    $_mac =~ tr/[A-Z]/[a-z]/; # convert MAC Address to lowercase

    warn "Lease Start: $_datetime\n" if $_datetime && $ENV{debug};  # Print if debugging
    warn "Lease End: $_lease_end\n" if $_lease_end && $ENV{debug};  # Print if debugging
    warn "IP: $_ip\n" if $_ip && $ENV{debug};  # Print if debugging
    warn "Mac: $_mac\n" if $_mac && $ENV{debug};  # Print if debugging

    if (/on.*\(([^).*]+)\) via/) { ## Grab the hostname if there is one
      $_wins = $1;
      $_wins =~ s/Hostname Unsuitable for Printing//; ## Remove this string if we see it
      $_wins = substr($_wins,0,47); ## Grab just 48 characters if it is long
      warn "WINS: $_wins\n" if $_wins && $ENV{debug};  # Print if debugging
    }

    $_lease_end =~ s/^(....)\/(..)\/(..):(.*)$/$1-$2-$3 $4/; # Convert slashes to dashes

    # Execute the query using bind parameters:
    $sth->execute($_datetime, $_lease_end, $_ip, $_mac, $_wins) or warn $DBI::errstr;
    $dbh->commit;
    $_dhcpcounter++;

  } ## End while-stdin
  $dbh->disconnect() or warn "Disconnection failed: $DBI::errstr\n";
  $_dhcpcounter =~ s/(^[-+]?\d+?(?=(?>(?:\d{3})+)(?!\d))|\G\d{3}(?=\d))/$1,/g;
  print "DHCP: $_dhcpcounter records imported to DHCP tables\n";
```

```
  return;
} # end DHCP log importing subroutine
```

Appendix B: Converting Time Zones

The following two Perl functions convert timestamps between Eastern and UTC. This code is written to work on a server whose time is Eastern and may perform differently if your server is set to UTC.

```
use Date::Format;
use Date::Parse;
use Time::Timezone;

#############################################################
## EasternToUTC
## - given a string in Eastern: YYYY-MM-DD HH:MI:SS
## - returns a string in UTC: YYYY-MM-DD HH:MI:SS
sub easternToUtc {
        my ($east) = @_;
        if (defined $east) {
                return time2str("%Y-%m-%d %H:%M:%S", str2time($east), 'UTC');
        } else {
                return;
        }
}

#############################################################
## UTCtoEastern
## - given a string in UTC: YYYY-MM-DD HH:MI:SS
## - returns a string in Eastern: YYYY-MM-DD HH:MI:SS
sub utcToEastern {
        my ($utc) = @_;
        if (defined $utc) {
                $utc .= ' UTC'; # Concatenate this to the end
                my $epoch = str2time($utc);
                return time2str("%Y-%m-%d %H:%M:%S", $epoch, tz2zone("EST5EDT", $epoch));
        } else {
                return;
        }
}
```

The following code is written to work on a server whose time is set to UTC and converts an Eastern timestamp to UTC:

```
my $_eastern = “2011-05-01 12:13:14”;
$ENV{TZ} = ':/usr/share/zoneinfo/US/East-Indiana';
my $_zone = tz2zone("EST5EDT", (str2time($_eastern)));
delete $ENV{TZ};
my $_utc_time = time2str("%Y-%m-%d %H:%M:%S", str2time($_eastern." $_zone"), 'UTC');
```

Appendix C: WHOIS Code

The following Perl code will discover abuse contacts for an IP address:

```
use Net::Whois::Raw;
use IP::Authority;
use IP::Country::Fast;

#####################################################
## Given an IP address, returns a whois server
sub getWhoisServer {
  my $ip = shift;
  ## IP::Authority takes an IP and returns the RIR two-char code
  my $reg = IP::Authority->new();
  my $auth = $reg->inet_atoauth($ip);
  unless ($auth) {
          return 0; # Generally these are ARIN, but this doesn't seem to matter
  } elsif ($auth eq 'AR'){
          return "whois.arin.net";
  } elsif ($auth eq 'RI') {
          return "whois.ripe.net";
  } elsif ($auth eq 'LA') {
          return "whois.lacnic.net";
  } elsif ($auth eq 'AP') {
          return "whois.apnic.net";
  } elsif ($auth eq 'AF') {
          return "whois.afrinic.net";
  } else {
          return 0;
  }
}
```

```
#####################################
## Whois IP to Abuse Contacts (emails)
## Given an IP address or network handle. Returns the email addresses for abuse notices.
sub whois_ip_to_abuse_contacts {
  my $input = shift;
  my $whois_server = getWhoisServer($input);
  print "Whois Server is: $whois_server\n" if $ENV{debug};
  my ($domain_info_arrayref, %abuse_contact, %all_contacts); # Prepare these values
  my $domain_info_str = "";
  unless ($input =~ m/NET/) { # Is this a network handle?
        # If not, run the whois query on the IP address:
    $domain_info_str = Net::Whois::Raw::whois($input, $whois_server);
  } else {
        warn "Skipping whois because input was NET-handle\n" if $ENV{debug};
  }

  ## If there is an email address in the domain information
  if ($domain_info_str =~ m/[\w\.\-]*\@[\w\.\-]*/) {
    my @temp = split("\n", $domain_info_str);
    $domain_info_arrayref = \@temp;
  } else {
    # No email address found yet. Keep searching:
    if ($whois_server) {
      $domain_info_arrayref =  Net::Whois::Raw::whois_query($input, $whois_server);
    } else {
          # If we can't determine the server, we just guess that it is ARIN
      $whois_server  = 'whois.arin.net';
      $domain_info_arrayref =  Net::Whois::Raw::whois_query($input, $whois_server);
    }
  }
  print "Domain info: " . Dumper($domain_info_arrayref) . "\n" if $ENV{debug};
  foreach my $line (@$domain_info_arrayref) {
    # Search each line for abuse email addresses
    if ($line =~ m/AbuseEmail:\s+([\w\@\.\-]*)\s*$/i) {
      $abuse_contact{$1}++;
    } elsif ($line =~ m/\s+([\w\.\-]*\@[\w\.\-]*)\s*$/i) {
          # Grab all email addresses
      $all_contacts{$1}++;
    }
  }

  if (%abuse_contact) { # If an abuse contact was found, it is preferred:
```

```
    return join(',', (keys %abuse_contact));
  } elsif (%all_contacts) { # Otherwise we use the other address(es) found:
    return join(',', (keys %all_contacts));
  } else { # No abuse contact found; look at the handle, if possible
    warn "No contact found; recursing\n" if $ENV{debug};
    foreach my $line (@$domain_info_arrayref) {
         # Check each line of the domain information for a network handle:
      if ($line =~ m/(NET[\d\-]+)\)/i) {
        my $handle = $1;
        warn "Handle is: $handle\n" if $ENV{debug};
               # Recurse using the network handle:
        return whois_ip_to_abuse_contacts($handle);
      }
    }
  }
}
```

Appendix D: Charts

This Perl function returns the HTML necessary to print a chart using the Google Chart Tool. More details about Google's API is available at:

http://code.google.com/apis/chart/

```
use CGI qw(:standard);
use Data::Dumper;
use Date::Format;
use Date::Parse;
use Math::Round qw(nearest_ceil nearest_floor);
use POSIX qw(ceil);

############################################################
## This returns the HTML to print the JavaScript and image
## Data is a reference to a hash: keys are series; values are hashes of dates and values.
## Data expected to look something like this sample hash:
## {
##  'Series1' => {'2011-05-01' => 28,
##                '2011-06-01' => 33,
##                '2011-07-01' => 23,
##                '2011-08-01' => 30,
##                '2011-09-01' => 29,
##                       },
```

```
##  'Series2' => {'2011-05-01' => 99,
##                '2011-06-01' => 103,
##                '2011-07-01' => 87,
##                '2011-08-01' => 92,
##                '2011-09-01' => 87,
##                       },
## };
## Size is either small, medium, or large.
## group_by is: calendar year, academic year, quarter, month, week, day or none.
## label_order (optional) is the order the series should be listed.
sub line_chart {
  my ($data, $size, $title, $group_by, $label_order) = @_;
  unless ($label_order) { # If label_order undefined, create that array ref
    my @lab = (keys %$data);
    $label_order = \@lab;
  }
  # Next we re-sort the hash based on the dates:
  my %date_hash;
  foreach my $label (@$label_order) {
    foreach my $date (keys %{$data->{$label}}) {
      $date_hash{$date}{$label} = $data->{$label}->{$date};
    }
  }
  # Start printing the JavaScript:
  my $return = "<script type='text/javascript'
src='http://www.google.com/jsapi'></script>
               <script type='text/javascript'>
                     google.load('visualization', 1, {packages:['corechart']});\n
                     google.setOnLoadCallback(drawLineChart);\n
                     function drawLineChart() {
                       var data = new google.visualization.DataTable();\n
                       data.addColumn('string', 'Year');\n";

  foreach my $label (@$label_order) { $return .= "data.addColumn('number', '$label');\n";
}
    foreach my $date (sort (keys %date_hash)) {
      my $date_string = $date;
      if ($group_by eq 'calendar year') {
        $date_string = time2str('%Y', str2time($date));
      } elsif ($group_by eq 'academic year') {
        $date_string = time2str('%y', str2time($date)) . '-' . time2str('%y',
(str2time($date) + 31557600)); # String like 08-09
      } elsif ($group_by eq 'quarter') {
```

```
        $date_string = 'Q' . time2str('%q', str2time($date));
        $date_string = time2str('%Y', str2time($date)) if ($date =~ m/-01-01$/);
      } elsif ($group_by eq 'month') {
        $date_string = time2str('%b', str2time($date));
        # Print year instead of January:
        $date_string = time2str('%Y', str2time($date)) if ($date_string eq 'Jan');
      } elsif ($group_by eq 'week') {
        $date_string = 'w' . time2str('%U', str2time($date));
        $date_string = time2str('%Y', str2time($date)) if (($date =~ m/-01-(..)$/) && ($1
<= '07'));
      } elsif ($group_by eq 'day') {
        $date_string = time2str('%a', str2time($date));
        $date_string =~ s/.$//; # Remove last character
      }
      $return .= "data.addRow(['$date_string', ";
      foreach (@$label_order) {
        next unless defined($date_hash{$date}{$_}); # Skip if no value is defined.
        $return .= $date_hash{$date}{$_} . ","
      }
      $return =~ s/,$//; # Remove last extra comma
      $return .= "]);";
    }
    #warn "Currently looks like $return\n"; # Debuggung
    $return .= "var chart = new
google.visualization.LineChart(document.getElementById('line_chart_div'));";
    $return .= "chart.draw(data, {curveType: 'none',";
    if ($size eq 'large') {
      $return .= "width: 1500, height: 480,";
    } elsif ($size eq 'medium') {
      $return .= "width: 1024, height: 320,";
    } elsif ($size eq 'small') {
      $return .= "width: 768, height: 320,";
    } elsif ($size eq '720p') {
      $return .= "width: 1280, height: 720,";
    }
    $return .= "lineWidth: 1,
                pointSize: 4,
                title: '$title',
                colors: ['#7d110c','#2d637f', '#c78036', '#6f679d', '#808a53']});
                }
                </script>
                <div id='line_chart_div'></div>\n\n";
```

```
    return $return; # Return the HTML to print the chart
}
```


AUTHOR * MERGEFORMAT Jonathan Sweeny, jsweeny@gmail.com