Kindsicher: Safe Internet Browsing for Unsupervised Children

Chaitanya Achan
University of Utah
cachan6460@gmail.com

Philip Lundrigan
University of Utah
philipbl@cs.utah.edu

Christian Schreiner
University of Utah
cas@cs.utah.edu

ABSTRACT
In a modern society, children must increasingly use the Internet for required tasks such as homework and communication with parents and other family members, not just for optional tasks (such as games and entertainment). The amount of required use means that children must often use the Internet when their parents cannot supervise them. This carries significant risk that the children will encounter Internet content they are not able to handle, or that would exploit them. Most previous work on improving children's Internet safety focuses on identifying "bad" content and blocking it. The amount of work to identify "bad" content is, however, beyond the capabilities of most families. Commercial services exist that classify content, but they are still stretched to adequately deal with the volume of legitimate content, and black hats are continually finding new means of circumventing the services' products. Further, outsourcing content classification prevents a parent from restricting content for family-specific reasons. For example, a child may have a phobia about spiders, or a parent may want to insist on accompanying a child whenever they visit certain online shopping sites, so the parent can teach good consumer practices during the shopping session.

We present Kindsicher, a children's Internet safety system that addresses these issues by taking the opposite approach: parents define the Internet sites they believe are appropriate for their children to visit unsupervised, and access to other sites requires parental intervention. Kindsicher has the additional advantage that it is implemented as home network infrastructure, so its protection is automatically extended to new devices brought into the home (for example, when a friend arrives with a Wi-Fi tablet).

Author Keywords
Parental Filter, Snort, TCP Reset Attack

INTRODUCTION
In a modern society, children must increasingly use the Internet for required tasks such as homework and communication with parents and other family members, not just for optional tasks (such as games and entertainment). The amount of required use means that children must often use the Internet when their parents cannot continually supervise them. This carries significant risk that the children will encounter Internet content they are not able to handle, or that would exploit them.

Most previous work on improving children's Internet safety focuses on identifying "bad" content and blocking it. Some of this work relies on autoclassification (e.g. Dans Guardian [1]), some on human classification (e.g. OpenDNS [7]), and some on a combination (e.g. NetNanny [6]). The amount of work to identify "bad" content is, however, beyond the capabilities of most families.

Commercial services exist that classify content [7, 6, 5, 3], but they are still stretched to adequately deal with the volume of legitimate content, and black hats are continually finding new means of circumventing the services' products. It is disappointing to observe a pattern in commercial product literature of focusing on features and testimonials, and omitting technical details. [7, 4, 5, 3] This prevents parents (or a technically inclined friend of the parents) from making a rational comparison between products and evaluating potential for ineffectiveness or circumvention without purchasing each product and performing detailed technical experiments. Further, outsourcing content classification prevents a parent from restricting content for family-specific reasons. For example, a child may have a phobia about spiders, or a parent may want to insist on accompanying a child whenever they visit certain online shopping sites, so the parent can teach good consumer practices during the shopping session.

This paper presents Kindsicher[1], a children's Internet safety system that addresses these issues by taking the opposite approach: parents define the Internet sites they believe are appropriate for their children to visit unsupervised, and access to other sites requires parental intervention. Kindsicher offers the following advantages:

- Kindsicher allows the parents full control over the Internet sites to be designated as "safe". Kindsicher's

[1]Kindsicher is a portmanteau of the German words for "child" and "safe."

architecture allows for parents to define exceptions to the approved site list so children can browse additional content when parents are present. It would be conceptually simple to extend this so parents could temporarily add sites via a web-capable smartphone, in case children needed to access another site while their parents were away.

- Kindsicher is completely implemented as part of a home network infrastructure, so its protection is automatically extended to new devices brought into the home (for example, when a friend arrives with a Wi-Fi tablet). Most other solutions have a component that must be installed on the client device.
- Since Kindsicher is completely implemented on the home network, it does not slow network performance while content is routed via a cloud-based filtering server. This can be a factor for current and modern rich-content web pages.
- Kindsicher logs all Internet traffic, so parents can have a rational conversation with their children if a problem occurs. (Problems might range from a teenager excessively playing online games, to a child coming to parents in tears because something seen on the Internet scared him/her.)
- Kindsicher's logging feature means that if black hat malware tried to attack the Kindsicher server, the attack would be logged, so technically competent humans could address the problem. Even a non-technically inclined parent could still see which computer in the home was involved in the attack (either as a source or as a victim), and take that computer to a repair shop for malware removal and an anti-virus update.
- Kindsicher blocks traffic by interrupting TCP connections, as opposed to blocking DNS name lookups, so it cannot be circumvented by hard-coding an IP address in a URL (e.g. http://155.98.65.24/), which is a common black-hat tactic.
- Kindsicher is completely open-source software, so the technically inclined are free to examine it, improve it if desired, and share their results with others.

No product or system exists today that has all of these features. We hope that by developing Kindsicher, we can show that each of these features is important and should be integrated into previous systems.

RELATED WORK

Protecting children from the potential harms of the Internet has been studied from a psychological point of view [21, 18]. Guidelines have been developed to help parents know how to expose their children to the Internet. For example, the EU has published articles on this subject [13, 16]. Other studies have looked at these types of guidelines to see if they are effective or not [15]. These studies show that there is a need for Internet filtering in homes to help protect children. These studies differ from our work in that they try to understand and measure the effect of the Internet on children's lives, whereas we look at building a filtering system.

There have also been studies on using firewalls to control Internet access [14, 12]. These approaches take a similar approach to our system, but in our system, we don't use a firewall or any hardware specific device. Also, in our system we decide that we don't want all traffic to travel through one node, like a firewall. This is described in more detail in the Design section.

A important part of our system is defining a correct whitelist. Whitelist implementations have been looked at to prevent specific web-based attacks [20, 11]. These whitelists were used for different purposes than what we are using our whitelist for.

Finally, prior work also includes web-page classification mechanisms [10, 22, 18] with the intent of identifying content and controlling access. We decided to avoid such techniques because they are hard to maintain and easily tricked. In our system, we do not classify webpages. Instead we block pages if they are not on a whitelist that was generated by the parents. This gives parents complete control over the system and as a result, parents are not dependent on the whims of the classification algorithm.

DESIGN

The goal of our system is two fold: block websites that are not part of the white list and to record network statistics about the Internet usage of the family. The second goal is important because effective parenting does not mean controlling children, but helping them learn to make good decisions themselves.[17] Regarding the Internet, this means helping the children develop healthy Internet usage habits. By logging Internet usage, parents can identify any unhealthy Internet behaviors. For example, say a teenager is staying up late to use the Internet and the website they are viewing is on the whitelist. The content the teenager is viewing might not be bad (such as a simple Internet game), but staying up late is not a healthy habit. With this information, the parents are able to address this problem in a direct manner and help the child learn from his/her mistake.

For both blocking and reporting, we used the intrusion detection system (IDS) Snort [8]. An IDS is a piece of software that sits on individual clients (called host based IDS or HIDS) or on a network (called network based IDS or NIDS). It monitors traffic that passes through it and detects if any type of attack is occurring. It records this information in a database and notifies an network administrator if it detects any serious problems.

The IDS' host can be installed in either of two topological configurations: in-line and stub. An in-line configuration requires the IDS host to forward every packet entering or leaving the network. It guarantees that the IDS will able to process (or block) every packet, at the expense

```
<rule_action> <protocol>
    <src_ip_addr> <src_port>
    <direction>
    <dst_ip_addr> <dst_port>
    (<meta_data>)
<rule_action>: alert, log, pass, activate,
    dynamic, drop, reject, sdrop
<protocol>: tcp, udp, ip, etc.
<direction>: ->, <-, <>
Example:
reject tcp
    any any
    <>
    10.0.0.10 80
    (sid: 1; msg: "rejecting!";)
```

Figure 1. The general syntax of a Snort rule with an example.

of slowing down network communication to the speed that the IDS host can handle. If the in-line IDS host becomes overloaded or fails, all network communication slows or stops. On the other hand, a stub-installed IDS host requires a network hub (*not* a switch). (Recall that a network hub sends all traffic it receives out on all its ports, unlike a switch, which determines each packet's destination host, and only sends the packet out on the appropriate port.) The hub allows the IDS to silently listen to all traffic passing through the hub, and if it determines a TCP packet is problematic, the hub allows the IDS to interrupt the packet's TCP connection by sending a TCP reset signal to the packet's source and destination hosts. If a stub-installed IDS becomes overloaded or fails, it allows some or all packets through without logging and/or interrupting them.

While passing unfiltered packets admittedly weaken the primary goals of Kindsicher, we opted for a stub-configured IDS host, on the theory that it is better to have weaker browsing protection than to risk children not being able to communicate in an emergency (e.g. using an Internet phone to call a parent after a sibling trips and breaks an arm). The network topology we expect is shown in Figure 2.

For Kindsicher, we used Snort as our IDS. Snort is rule based which means that a system administrator can write rules that cause Snort to perform an action whenever the rule's conditions are met. Figure 1 shows the general syntax of a rule. There are communities of Snort users that write rules for common attacks and network problems so that typical system administrators do not need a deep knowledge of intrusion detection to get benefit from Snort.

How Snort was used to accomplish the goals of blocking and reporting is outlined in the Blocking and Reporting sections.

Distinguishing Children from Parents

The TCP/IP protocol, and the HTTP protocol that rides on top of it, identify the sending computer, *NOT* the user name on that computer. Since Kindsicher's goal is to restrict access for children, not for adults, the only way to distinguish between children's Internet access and adults' access is to designate certain computers for children's use only, and other computers for adults' use only. Thus, there needs to be some mechanism to ensure that the children can only log into hosts designated for children.

The network we used for testing uses SNISR[2] for login directory services. The current released version, SNISR 1.1, allows any user to log in on any host. Part of this project, therefore, was upgrading SNISR to allow the network administrator to specify classes of hosts that only certain users could log into. SNISR 2.0's baseline ALPHA_6 contains this feature, and is expected to begin beta testing soon.

Blocking

Figure 1 shows the general syntax for a Snort rule. There are eight types of Snort rules: `alert`, `log`, `pass`, `active`, `dynamic`, `drop`, `reject`, and `sdrop`. Each type is described below:

- `alert`. `alert` is the most commonly used type of rule. It sends an alert to who ever is running the Snort server *and* logs the packet(s) for later inspection.
- `log`. `log` logs the packet that triggered the rule, but does not send an alert. This can be used for a rule that is not critical but the network administrator wants to be able to look at it later.
- `pass`. `pass` does not do anything with the packet. The packet is ignored and allowed to proceed to its destination. This is a way of turning off a rule that is not necessary.
- `active`. The is the same as `alert` except that after sending an alert, a dynamic rule is turned on.
- `dynamic`. A `dynamic` rule will do nothing until activated by an `active` rule. Once it has been activated, it acts as a `log` rule.
- `drop`. `drop` blocks the packet and logs it. This only works if Snort is in inline mode.
- `reject`. `reject` blocks the packet (if in inline mode), logs it, and sends a TCP reset message (for a TCP connection) or a ICMP port unreachable message (for UDP connection) to both ends of the connect.
- `sdrop`. Same as `drop` except that the packet is not logged.

[2]SNISR is a network login and directory information system, akin to LDAP, NIS, or Hesiod. Its signature advantage is that all clients have a local record of account and group information at all times, so mobile clients can disconnect from the network and reconnect at will. See http://www.mathoni.net/cas/swforge/snisr for more information.

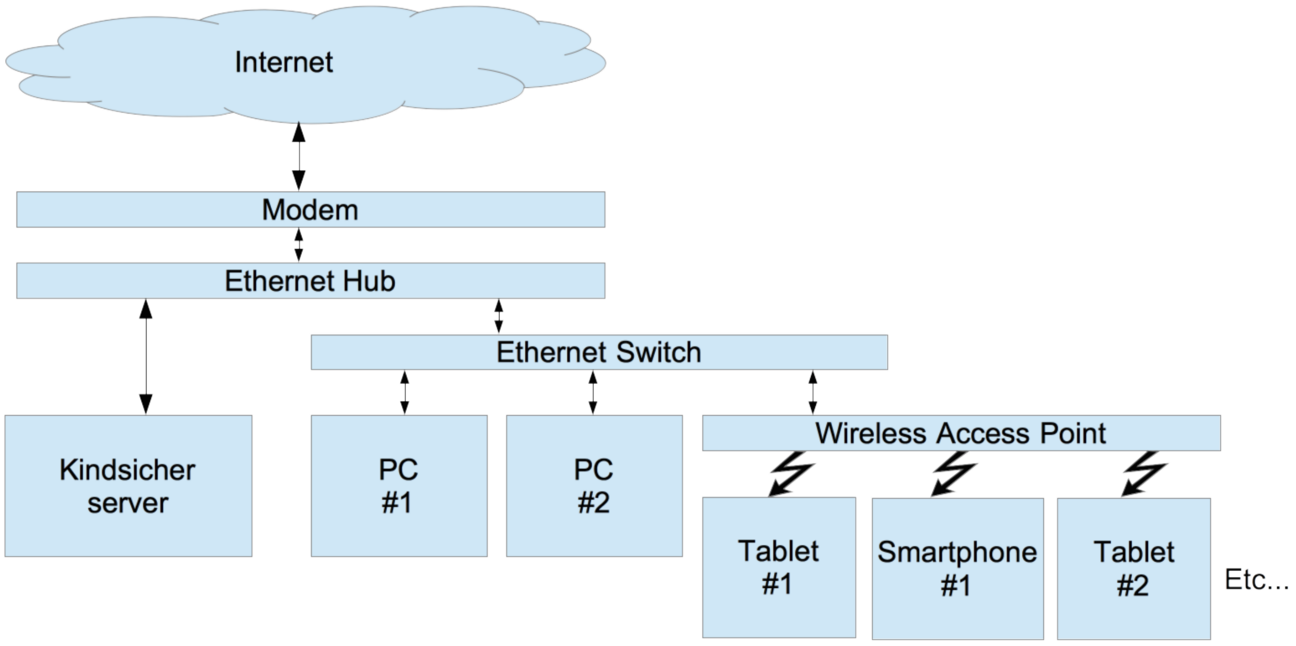


Figure 2. The network topology of the Kindsicher system. An Ethernet hub is used to broadcast all traffic from the Ethernet switch to the Kindsicher server. The Ethernet switch contains a mixture of PCs and wireless clients. Any device that connects to the Ethernet switch will be automatically protected by the Kindsicher server.

For the purposes of blocking websites, we are interested in `drop`, `reject`, and `sdrop`. We want to be record when a website is being blocked (we want the packet logged), so `sdrop` does not fit our needs. For `drop` to work correctly, Snort needs to configured in inline mode. Inline mode is when the Snort server is inline with Internet traffic. For example, the Snort server would be between the local network and the Internet connection. All data is inspected by the Snort server and either blocked or passed through. As our design section describes, this configuration has important implications. This means that if the Snort server fails then the Internet connection will be lost. This also means that the Snort server could potentially act as a bottle neck, slowing down traffic. Because of these reasons, we decided that putting Snort inline in the network did not fit our needs. This means that `block` is not a valid action.

The last rule action is reject. Since we are concerned about blocking HTTP requests, we concern ourselves with only the TCP functionality of reject and ignore the UDP functionality. Reject uses a well studied TCP attack called TCP Reset attack [19].

For a TCP Reset attack to work, there must be a man in the middle. It monitors the TCP connections that are being made between the network and the Internet. When the man in the middle wants to stop a connection, it sends a TCP RST packet to both ends of the connections. Both ends of the connection think that the packet came from the other end so it honors the RST packet. For this to work, the man in the middle must know the sequence number and ACK number of the TCP stream. This can easily be found by looking at the header of TCP packets. Once both ends of the connect receive the TCP RST packet, they stop sending data and close the connection. Figure 3 shows a packet flow of a TCP reset attack.

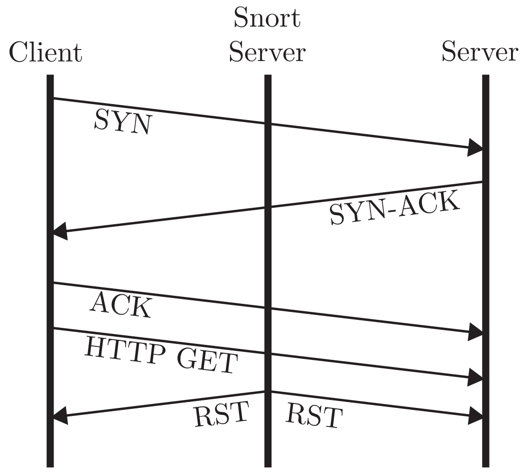
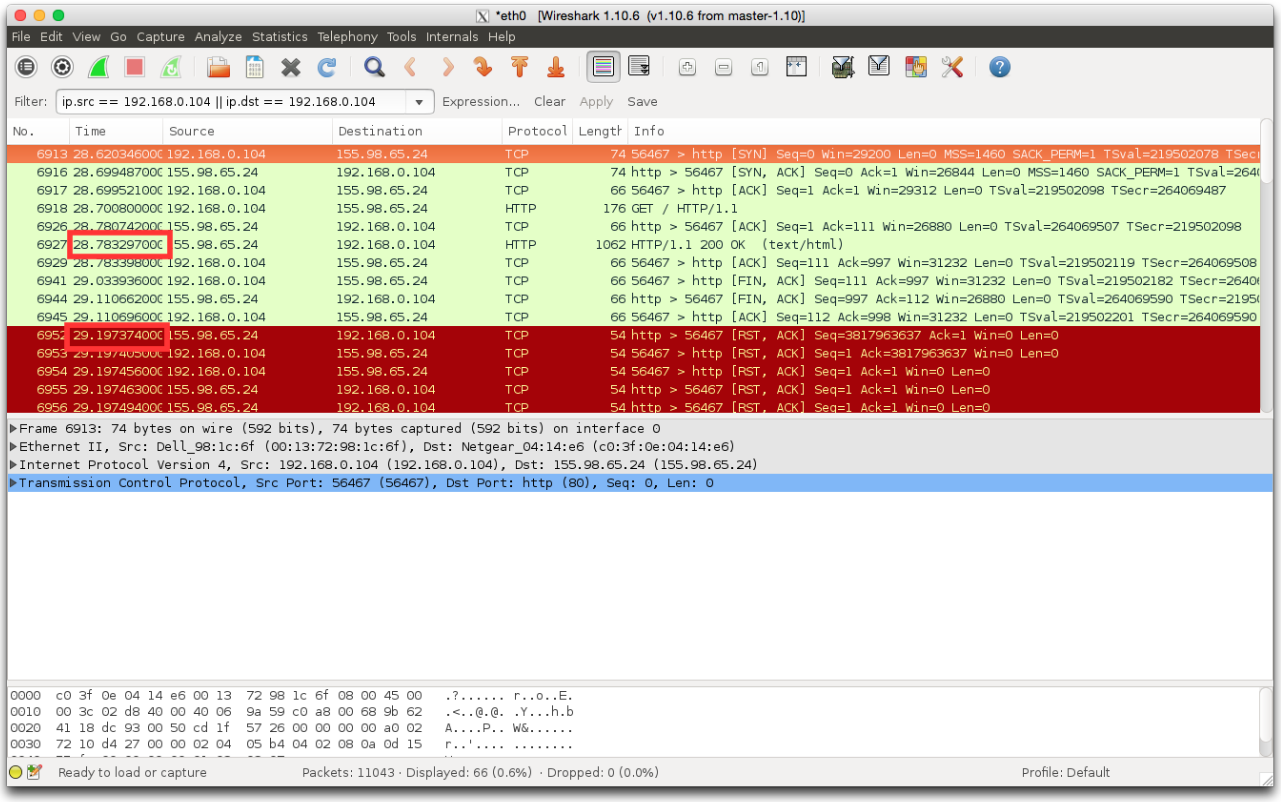


Figure 3. The flow of how a TCP RST attack works in our system. When the Snort server detects a HTTP GET request, the Snort server sends a TCP RST packet to both ends of the connection (Client and Server).


*eth0 [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: ip.src == 192.168.0.104 || ip.dst == 192.168.0.104
Expression... Clear Apply Save

```
No.   Time         Source          Destination    Protocol Length Info
 6913 28.62034600C 192.168.0.104   155.98.65.24   TCP        74 56467 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=219502078 TSec
 6916 28.69948700C 155.98.65.24    192.168.0.104  TCP        74 http > 56467 [SYN, ACK] Seq=0 Ack=1 Win=26844 Len=0 MSS=1460 SACK_PERM=1 TSval=264
 6917 28.69952100C 192.168.0.104   155.98.65.24   TCP        66 56467 > http [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=219502098 TSecr=264069487
 6918 28.70080000C 192.168.0.104   155.98.65.24   HTTP      176 GET / HTTP/1.1
 6926 28.78074200C 155.98.65.24    192.168.0.104  TCP        66 http > 56467 [ACK] Seq=1 Ack=111 Win=26880 Len=0 TSval=264069507 TSecr=219502098
 6927 28.78329700C 155.98.65.24    192.168.0.104  HTTP     1062 HTTP/1.1 200 OK  (text/html)
 6929 28.78339800C 192.168.0.104   155.98.65.24   TCP        66 56467 > http [ACK] Seq=111 Ack=997 Win=31232 Len=0 TSval=219502119 TSecr=264069508
 6941 29.03393600C 192.168.0.104   155.98.65.24   TCP        66 56467 > http [FIN, ACK] Seq=111 Ack=997 Win=31232 Len=0 TSval=219502182 TSecr=2640
 6944 29.11066200C 155.98.65.24    192.168.0.104  TCP        66 http > 56467 [FIN, ACK] Seq=997 Ack=112 Win=26880 Len=0 TSval=264069590 TSecr=2195
 6945 29.11069600C 192.168.0.104   155.98.65.24   TCP        66 56467 > http [ACK] Seq=112 Ack=998 Win=31232 Len=0 TSval=219502201 TSecr=264069590
 6952 29.19737400C 155.98.65.24    192.168.0.104  TCP        54 http > 56467 [RST, ACK] Seq=3817963637 Ack=1 Win=0 Len=0
 6953 29.19740500C 192.168.0.104   155.98.65.24   TCP        54 56467 > http [RST, ACK] Seq=1 Ack=3817963637 Win=0 Len=0
 6954 29.19745600C 192.168.0.104   155.98.65.24   TCP        54 56467 > http [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 6955 29.19746300C 155.98.65.24    192.168.0.104  TCP        54 http > 56467 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
 6956 29.19749400C 155.98.65.24    192.168.0.104  TCP        54 http > 56467 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

▶Frame 6913: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
▶Ethernet II, Src: Dell_98:1c:6f (00:13:72:98:1c:6f), Dst: Netgear_04:14:e6 (c0:3f:0e:04:14:e6)
▶Internet Protocol Version 4, Src: 192.168.0.104 (192.168.0.104), Dst: 155.98.65.24 (155.98.65.24)
▶Transmission Control Protocol, Src Port: 56467 (56467), Dst Port: http (80), Seq: 0, Len: 0

0000  c0 3f 0e 04 14 e6 00 13  72 98 1c 6f 08 00 45 00   .?...... r..o..E.
0010  00 3c 02 d8 40 00 40 06  9a 59 c0 a8 00 68 9b 62   .<..@.@. .Y...h.b
0020  41 18 dc 93 00 50 cd 1f  57 26 00 00 00 00 a0 02   A....P.. W&......
0030  72 10 d4 27 00 00 02 04  05 b4 04 02 08 0a 0d 15   r..'.... ........

Ready to load or capture        Packets: 11043 · Displayed: 66 (0.6%) · Dropped: 0 (0.0%)        Profile: Default
```

Figure 4. A screenshot of a Wireshark trace. In this trace, the Snort server is .41 seconds too late in sending TCP RST packets.

In the case of our system, the Snort server is the man in the middle. When a reject rule is triggered, the Snort server sends a TCP RST packet to each end of the connection.

Snort's reject rule works in theory, but in practice it has some problems. An inherent issue with not having our Snort server inline but instead having it rejecting connections is that there could be conditions such that a TCP connection could finish transmitting all of its data before the Snort server has time to respond. This is exactly what happens under certain conditions. Figure 4 shows a screenshot of a Wireshark trace. In it, the TCP connection is established, data is sent, and then the Snort server sends its TCP RST packets. The time between when the data is received by the client and when the Snort server sends a TCP RST packet is .41 seconds.

To really understand the problem with Snort being slow, we ran Snort in many different environments to make sure it wasn't a platform or topology specific problem. We ran the server on a MacBook Pro running OS X 10.10, a home network server running Ubuntu 14.04, and on an Emulab [2] node running Ubuntu 14.04. In the first two environments, the same problems occurred – Snort would not react fast enough to block all TCP traffic. In the third environment, we couldn't test a web server on the Internet (because of how Emulab is set up), but Snort was able to block a web server that was part of the Emulab experiment.

After some experimentation, there seems to be two factors that play a role in if the Snort server is able to reject a TCP connection or not. The first is the latency involved in making the request. If the request is being made to a server that is near by geographically, the Snort server has a harder time blocking it because the connection is so fast. The second factor is how big the web page that is being downloaded. In our tests, we found that Snort was unable to block web pages that could be contained inside one TCP packet. However, if web page was large enough that there needed to be a few packets to send the data, then Snort could block it most of the time. This was also true with large images as well.

To try to minimize this problem, we tried reducing the functionality of Snort. Snort is a IDS and has a lot more functionality that we don't need to block websites. By reducing what Snort is loading in its configuration, we can speed up the response time of the TCP RST packets. This helped to reduce the problem, but it did not fix it. In the future, we plan on looking at the source code of Snort and pulling out only the necessary parts to block websites.

GET / HTTP/1.1
host: www.nytimes.com

Results in HTTP GETs for:

a1.nyt.com	graphics8.nytimes.com
typeface.nytimes.com	pbs.twimg.com
js.nyt.com	rapidssl-ocsp.geotrust.com
int.nyt.com	js-agent.newrelic.com
p.typekit.net	core.fabrik.nytimes
markets.on.nytimes.com	ocsp.digicert.com

Figure 5. One HTTP GET request results in many more HTTP GET requests. Some of these requests are to different domains.

Creating the Whitelist

Part of being able to block websites adequately, is being able to produce a good list of URLs that should not be blocked (a whitelist). This is harder than it first sounds. This is in part due to the complexities of the Internet. Typically, when a web page is visited, resources from many different locations are loaded. An example of this is shown in Figure 5. This makes creating a realistic whitelist much more difficult. For example, a URL might be added to the whitelist, but for all of the websites important resources (images, fonts, etc.), other URLs need to be added to the whitelist as well. It becomes a lot harder as a parent to know what should be added to the whitelist.

To understand what a whitelist might look like for a typical family, one of the authors allowed for their families Internet traffic to be logged. This was done by using `tshark` [9] (a terminal version of Wireshark) to monitor Internet traffic and output the host name of a HTTP GET request to a file. This gave us insight into what kind of considerations need to be made when building an accurate whitelist.

Reporting

The second goal of this project is to provide parents with the ability to monitor and derive reports on their child's Internet access. We looked at incorporating BASE (Basic Analysis and Security Engine) into Kindsicher to render this functionality. BASE is based of the ACID (Analysis Console for Intrusion Databases) project and provides a web-based front-end to query and analyze alerts generated by the SNORT IDS system.

We first define a rule within SNORT to capture all HTTP traffic originating from the home network.

```
alert tcp any any <> any 80 (msg:"HTTP Alert")
```

This causes SNORT to log all HTTP requests from any source IP and port to any destination IP and port 80 (indicating HTTP) to a MySQL database. BASE then reads the alert information from this database and displays it in a web-browser. Figure 6 shows this interaction.

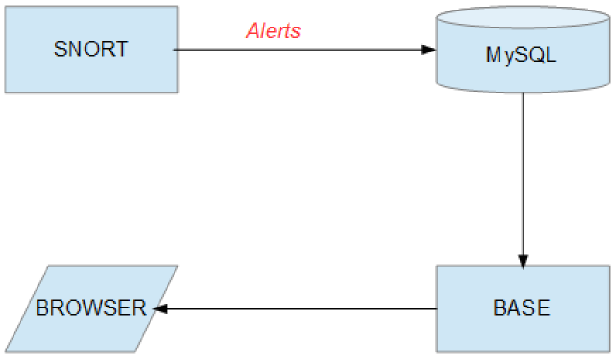


Figure 6. The flow of data for BASE. Snort stores alerts into a database which BASE reads from. Base is accessed through a web browser.

To access base you need to open a browser window and type in `<base server name>/base` in the URL field. Figure 7 shows the BASE home page presented to an user on logging into BASE.

For this project we were particularly interested in destination IP addresses. On clicking on the count of destination IP addresses shown in Figure 7, BASE displays a list of all the IP addresses accessed from the home network (Figure 8), along with other associated information such as the number of times the IP was accessed and the number of source addresses that accessed it.

BASE also provides several canned reports to graphically represent network data. Figure 9 shows an example of report selection and parameter specification.

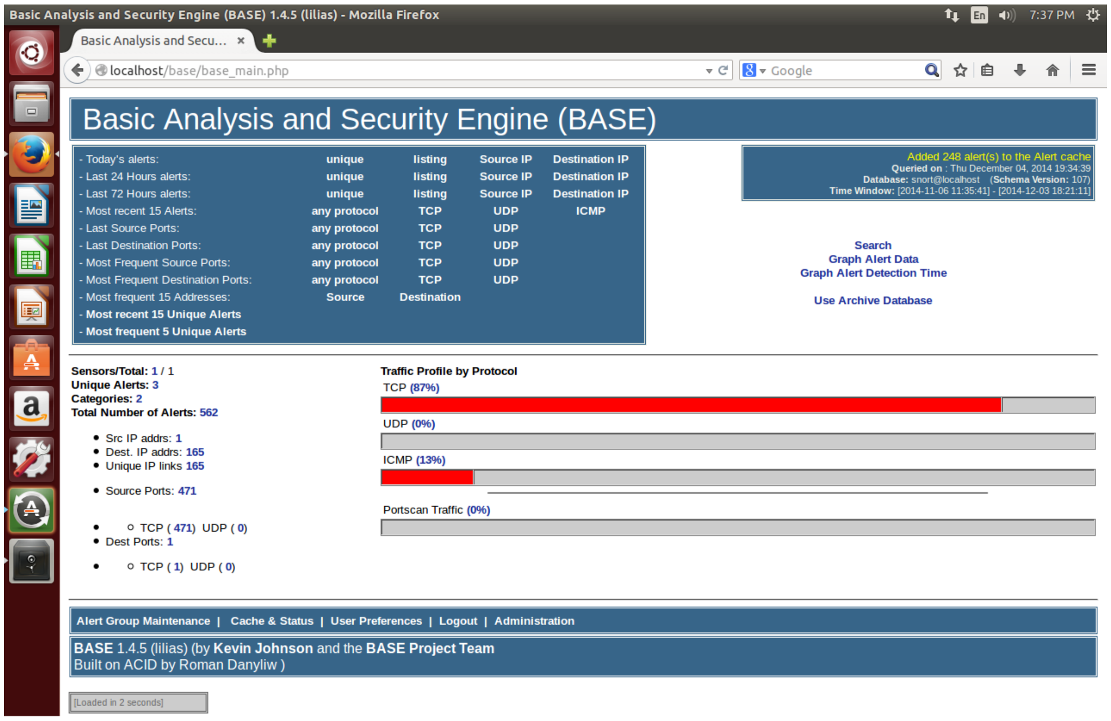
Based on our study of BASE we came up with the following conclusions.

Advantages of BASE:

- It is built to work in conjunction with SNORT.
- It provides a simple web-interface for users to view and analyze network traffic.
- It comes with several in-built reports to help in the analysis.
- Its open-source code is a useful starting point for customizations.

Disadvantages of BASE:

- It deals with only IP addresses and does not display the corresponding domain names. IP addresses are not intuitive to users and this raises a real concern given the fact that our target users are parents who are not necessarily tech-savvy.
- It also displays only the number of times a site has been accessed and gives no other information like the date and time of access.
- The in-built reports do not support any correlation of which source IP(s) accessed which destination IP(s),



Basic Analysis and Security Engine (BASE) 1.4.5 (lilias) - Mozilla Firefox

localhost/base/base_main.php

Basic Analysis and Security Engine (BASE)

- Today's alerts:	unique	listing	Source IP	Destination IP
- Last 24 Hours alerts:	unique	listing	Source IP	Destination IP
- Last 72 Hours alerts:	unique	listing	Source IP	Destination IP
- Most recent 15 Alerts:	any protocol	TCP	UDP	ICMP
- Last Source Ports:	any protocol	TCP	UDP	
- Last Destination Ports:	any protocol	TCP	UDP	
- Most Frequent Source Ports:	any protocol	TCP	UDP	
- Most Frequent Destination Ports:	any protocol	TCP	UDP	
- Most frequent 15 Addresses:	Source	Destination		
- **Most recent 15 Unique Alerts**				
- **Most frequent 5 Unique Alerts**				

Added 248 alert(s) to the Alert cache
Queried on : Thu December 04, 2014 19:34:39
Database: snort@localhost (**Schema Version:** 107)
Time Window: [2014-11-06 11:35:41] - [2014-12-03 18:21:11]

Search
Graph Alert Data
Graph Alert Detection Time

Use Archive Database

Sensors/Total: 1 / 1
Unique Alerts: 3
Categories: 2
Total Number of Alerts: 562

- Src IP addrs: 1
- Dest. IP addrs: 165
- Unique IP links 165

- Source Ports: 471
  - TCP (471) UDP (0)
- Dest Ports: 1
  - TCP (1) UDP (0)

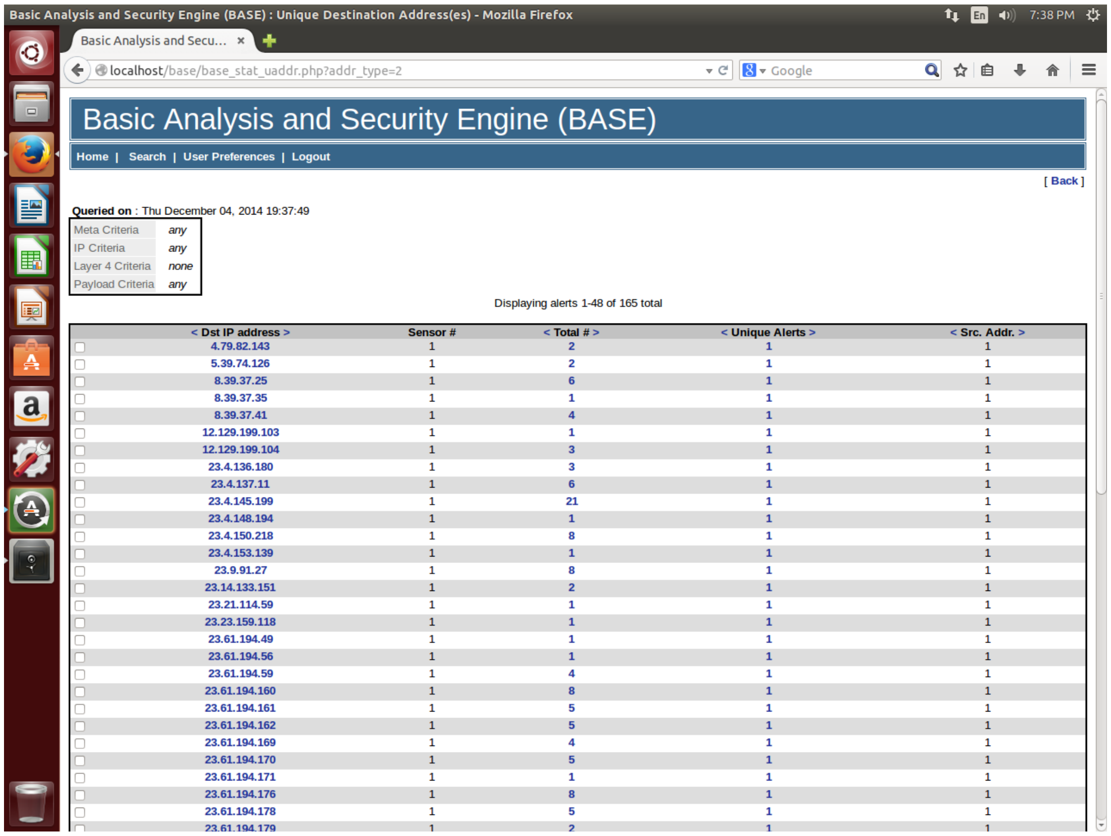
Traffic Profile by Protocol
TCP (87%)
UDP (0%)
ICMP (13%)
Portscan Traffic (0%)

Alert Group Maintenance | Cache & Status | User Preferences | Logout | Administration

BASE 1.4.5 (lilias) (by **Kevin Johnson** and the **BASE Project Team**
Built on ACID by Roman Danyliw)

[Loaded in 2 seconds]

Figure 7. A screenshot of the BASE home page.



Basic Analysis and Security Engine (BASE) : Unique Destination Address(es) - Mozilla Firefox

localhost/base/base_stat_uaddr.php?addr_type=2

Basic Analysis and Security Engine (BASE)

Home | Search | User Preferences | Logout

[Back]

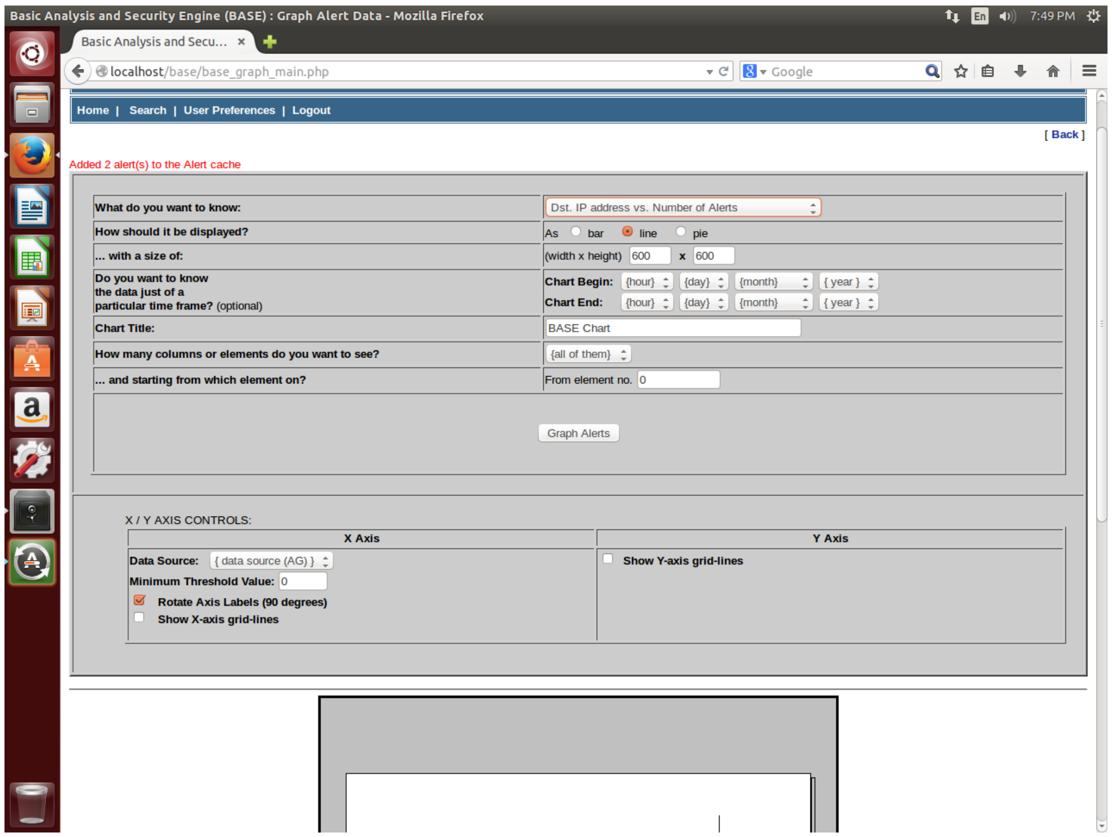
Queried on : Thu December 04, 2014 19:37:49

Meta Criteria	*any*
IP Criteria	*any*
Layer 4 Criteria	*none*
Payload Criteria	*any*

Displaying alerts 1-48 of 165 total

< Dst IP address >	Sensor #	< Total # >	< Unique Alerts >	< Src. Addr. >
4.79.82.143	1	2	1	1
5.39.74.126	1	2	1	1
8.39.37.25	1	6	1	1
8.39.37.35	1	1	1	1
8.39.37.41	1	4	1	1
12.129.199.103	1	1	1	1
12.129.199.104	1	3	1	1
23.4.136.180	1	3	1	1
23.4.137.11	1	6	1	1
23.4.145.199	1	21	1	1
23.4.148.194	1	1	1	1
23.4.150.218	1	8	1	1
23.4.153.139	1	1	1	1
23.9.91.27	1	8	1	1
23.14.133.151	1	2	1	1
23.21.114.59	1	1	1	1
23.23.159.118	1	1	1	1
23.61.194.49	1	1	1	1
23.61.194.56	1	1	1	1
23.61.194.59	1	4	1	1
23.61.194.160	1	8	1	1
23.61.194.161	1	5	1	1
23.61.194.162	1	5	1	1
23.61.194.169	1	4	1	1
23.61.194.170	1	5	1	1
23.61.194.171	1	1	1	1
23.61.194.176	1	8	1	1
23.61.194.178	1	5	1	1
23.61.194.179	1	2	1	1

Figure 8. List of destination IPs accessed from the network as shown by BASE.


Basic Analysis and Security Engine (BASE) : Graph Alert Data - Mozilla Firefox

localhost/base/base_graph_main.php

Home | Search | User Preferences | Logout

[Back]

Added 2 alert(s) to the Alert cache

What do you want to know:	Dst. IP address vs. Number of Alerts
How should it be displayed?	As bar [✓ line] pie
... with a size of:	(width x height) 600 x 600

Do you want to know the data just of a particular time frame? (optional)	Chart Begin: {hour} {day} {month} { year }
	Chart End: {hour} {day} {month} { year }

Chart Title:	BASE Chart
How many columns or elements do you want to see?	{all of them}
... and starting from which element on?	From element no. 0

Graph Alerts

X / Y AXIS CONTROLS:

X Axis	Y Axis
Data Source: { data source (AG) } Minimum Threshold Value: 0 [✓ Rotate Axis Labels (90 degrees)] Show X-axis grid-lines	Show Y-axis grid-lines

Figure 9. BASE report selection and initiation.

reducing a parent's understanding of their child's Internet access.

Although BASE does provide an avenue for users to view and analyze web access, it only partially meets the reporting goals for Kindsicher because of these disadvantages. This is further detailed in the Future Work/Improvements section.

FUTURE WORK

As described in the blocking section, we encountered some latency issues with Snort that inhibited its blocking ability under certain conditions. For Kindsicher to be effective in all circumstances, we need to investigate Snort further. This might involve modifying the source of Snort to remove all extra functionality. This could also involve writing our own program that blocks using TCP Resets.

In the Reporting section, we described some of the shortcomings of BASE, primary of which being its reporting on network access using only IP addresses. The reporting can be made more user friendly by incorporating domain names instead of or in addition to IP addresses. It could also be modified to provide a richer set of data to parents, that makes understanding their child's web access footprint much more intuitive.

Over the longer term, we also need a process that automates maintaining the addresses in the Snort white-list. We envision that a child can request addition of a new website domain as a text message to the parent's phone and the parent could then chose to approve or deny that request, automatically updating the white-list as needed. This would allow parents to change the whitelist with minimal effort.

CONCLUSION

We present a system called Kindsicher that provides an unique approach to protecting children from the potential harms of the Internet. Our system has two major goals: block all access to websites, except for sites that are on a white list and provide insight into how families are using the Internet through reporting. We describe each of the components to our system and how they fit together.

Over the course of designing the system, we found that building such a system is much harder than first assumed. The Internet is a complex place which makes it hard to build a system like this that is robust enough to withstand real usage. At the same time, that very complexity makes it ever so much more important for children to learn how to handle the advantages and risks that the Internet offers. A fully functional system implementing the principles of Kindsicher would be a useful tool to help parents teach their children how to deal with it.

REFERENCES

1. Dansguardian: true web content filtering for all. `http://dansguardian.org`.
2. Emulab: Total network testbed. `https://www.emulab.net`.
3. K9 web protection. `http://www1.k9webprotection.com`.
4. Kidlogger - freeware and open source parental controls for windows, mac, android. `http://kidlogger.net`.
5. Mcafee family protection. `http://home.mcafee.com/Store/PackageDetail.aspx?pkgid=342`.
6. Net nanny: We protect families. `http://www.netnanny.com`.
7. Opendns. `http://www.opendns.com`.
8. Snort. `https://www.snort.org`.
9. Tshark. `https://www.wireshark.org/docs/man-pages/tshark.html`.
10. Eda Baykan, Monika Henzinger, Ludmila Marian, and Ingmar Weber. A comprehehsive study of features and algorithms for url-based topic clasification. *ACM Transactions on the Web 5*, 3 (July 2011).
11. Genta Iha and Hiroshi Doi. An implementation fo the binding mechanism in the web browser for preventing xss attacks: Introducing the bind-value headers. In *2009 International Conference on Avaialbility, Reliability and Security*, IEEE Computer Society (2009), 966.
12. Hai Thanh Nguyen, et al. Enhanceing the effectiveness of web application firewalls by generic feature selection. via Oxford University Press, August 2012.
13. Holloway, D., G. L., and Livingstone, S. Zero to eight. young children and their internet use.
14. Ivan Ivanovic. Distribution of web traffic toward the centralized cloud firewall system. *RoEduNet International Conference 12th Edition* (2013).
15. Livingstone, S., and Helsper, E. J. Parental mediation of children's internet use. *Journal of broadcasting & electronic media 52*, 4 (2008), 581–599.
16. Livingstone, S., Haddon, L., Grzig, A. and lafsson, K. Risks and safety on the internet: the uk report.
17. Severe, S. *How to Behave So Your Children Will, Too!* Penguin Group, 2000, ch. 32, p. 246.
18. Wai H. Ho and Paul A. Watters. Statistical and structural approaches to filtering internet pornography. *2004 IEEE Internal Conference on Systems, Man and Cybernetics* (2004), 4792.
19. Watson, P. Slipping in the window: Tcp reset attacks. *Presentation at* (2004).
20. Weili Han, Ye Cao, Elisa Bertino, and Jianming Yong. Using automated individual white-list to protect web digital identities. *Export Systems with Applications 39*, 15 (Nov 2012), 11861–11869.
21. Ybarra, M. L., and Mitchell, K. J. Exposure to internet pornography among children and adolescents: A national survey. *CyberPsychology & Behavior 8*, 5 (2005), 473–486.
22. Zhaoyao Chen, Ou Wu, Mingliang Zhu, and Weiming Hu. A novel web page fitlering system by combining texts and images. *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence* (2006).