A Day in the Life Of a Web Application

FIRST 2010

Ken van Wyk (ken@krvw.com)
Scott McIntyre (scott@xs4all.net)

Quiz

Here's the scenario

- You've worked with your software dev team, business team, etc., to deploy a new web application
  - It passed various penetration tests and reviews
  - It is in production
- A major incident hits, and you find out the new app is hemorrhaging sensitive customer information
- The (unhappy) CEO calls on you, the CSIRT, to "fix things NOW!"

Now what?

1. What are the your top priorities?
2. How effective is the web app software at helping you do your job?
   - You did look over the logging capabilities in your security reviews, didn't you?

Here's my answer to #1

- Highest priority is to determine the business impact
- Second highest is situational awareness throughout the incident
- Third is to recommend a course of action to take, and then to coordinate execution of that plan

How about #2?

- Chances are you'll have plenty of logging to pore through
- Chances are most of it will be useless to your mission priorities

In reality

In my 20+ years of incident response experience:

- The CSIRT is called up after the fact
  - Or we find out about the incident by accident!
- All too often, the attackers have come and gone
- We have to assemble the puzzle from available data
  - There's never enough−or the right−available data

CSIRT needs to…

Determine the who, what, when, where, and how (WWWWH)

- Using existing records of the events
- Disk and network forensics often not terribly useful
  - After the fact may be too late
  - Time consuming and costly
  - Only used to find specific data
- Where do we look?
  - Logs, gobs of logs
  - OS logs, firewall logs, web server logs, but how about app?

Event logging

In a production data processing environment, there can be many sources of log data

- With luck, they're sent to a central log concentrator
- Consider the per-source perspective
  - What did the (router, firewall, web server, Java container, database) see?
  - Now, what did they report?
  - How do they speak to WWWWH?

The trouble with logs

Most are simply extensions of debugging hooks in code

- They are written for the *developer*, not the CSIRT
- Wrong audience and purpose

Let's take a look at a couple examples

Company background

- Large ISP
  - Range of services: DSL, VoIP, hosting
- Massive enterprise application infrastructure
  - Much of it is exposed to the big bad internet
- Security is taken very seriously
  - Availability is primary concern
  - Fraud prevention is close second

Examples −1

Does this mean anything (useful) to you?

```
Feb 11 09:09:30 server1 setuid-wrapper[76686]: zzz called "d/usr/bin/setuid/
atmping --monitor LINEIDNUMBER"

Feb 11 09:09:33 server1 setuid-wrapper[76727]: zzz called "d/usr/bin/setuid/
greplog --tail --appstream account=wibble1,"

Feb 11 09:09:34 server1 setuid-wrapper[76736]: zzz called "?^C/usr/bin/setuid/
zoeklog -i 1.2.4.4"

Feb 11 09:09:35 server1 greplog[76727]: zzz searches with TRACE for
account=wibble1,
```


On the surface this looks great. We have various ways to search for a customer's line activity in our RADIUS system, and since in Europe an IP address is considered personally identifiable information, and as such, covered under our data protection acts, we like to know if one of our employees is digging around into the activity of our customers.

The scripts which do the work are all running through a setuid wrapper which keeps important database passwords and other credentials out of the eyesight of the helpdesk, but, still lets them do their work.

So, what's the failure here?

The way this was written is that it ONLY reports user "zzz" as the person invoking it. I've changed their name, of course, but, the issue here is that the log looks meaningful from the point of view of logging something which may be a violation of privacy law, but, in reality, it is only logging that *someone* ran the command, but not actually *who*...

Examples −2

How about this one?

```
Feb  6 10:28:08 service8 ServiceCentre-1.36.24[9231]: [STATS] {Service:217} servicecentre_login:
username_mangled=54894ef40dca18a5

Feb  6 10:28:27 service6 ServiceCentre-1.36.24[2919]: [STATS] {Service:217} servicecentre_login:
username_mangled=bf2808e01aeb8deb

Feb  6 10:28:35 service6 ServiceCentre-1.36.24[2923]: [STATS] {Service:217} servicecentre_login:
username_mangled=8ec5c711d167964d

Feb  6 10:29:27 service6 ServiceCentre-1.36.24[2934]: [STATS] {Service:217} servicecentre_login:
username_mangled=2b82db4bbf54f7aa

Feb  6 10:29:41 service8 ServiceCentre-1.36.24[9258]: [STATS] {Service:217} servicecentre_login:
username_mangled=64aa378b8f32905c

Feb  6 10:29:55 service2 ServiceCentre-1.36.24[98842]: [STATS] {Service:217} servicecentre_login:
username_mangled=aa9e261c1f797bc5

Feb  6 10:29:57 service8 ServiceCentre-1.36.24[9263]: [STATS] {Service:217} servicecentre_login:
username_mangled=54894ef40dca18a5
```


These logs were made by developers. It's obvious. The system is behind load balancers, so the syslog log contains the actual hostname that generated the log and then the *exact* version number of the application itself, as well as the PID of the process that generated the log, and, the *line number* that generated it. But then the only thing that I get? A mangled MD5 hash of the customer's login identity, and a internally known "secret" value.

This means that I can go and look for a specific customer, by re-creating that hash, but, even if I do, there's nothing of any use here at all. I don't know the IP address, I don't know what they did, I don't have anything to go on here that is of use for security or auditing purposes. Just that a credential was used and what lines of code logged that use. Argh!

Examples −3

And what does this one tell you?

```
64.4.8.137 - - [24/Jan/2007:06:15:09 -0500] "GET /robots.txt HTTP/1.0" 200 0 "-"
"msnbot/1.0 (+http://search.msn.com/msnbot.htm)"

64.4.8.137 - - [24/Jan/2007:06:15:09 -0500] "GET /rss.xml HTTP/1.0" 200 8613 "-"
"msnbot/1.0 (+http://search.msn.com/msnbot.htm)"

64.4.8.137 - - [24/Jan/2007:06:38:41 -0500] "GET /robots.txt HTTP/1.0" 200 0 "-"
"msnbot/1.0 (+http://search.msn.com/msnbot.htm)"

64.4.8.137 - - [24/Jan/2007:06:38:41 -0500] "GET /about.php HTTP/1.0" 200 9770 "-"
"msnbot/1.0 (+http://search.msn.com/msnbot.htm)"

64.4.8.137 - - [24/Jan/2007:06:42:21 -0500] "GET /index.php HTTP/1.0" 200 5080 "-"
"msnbot/1.0 (+http://search.msn.com/msnbot.htm)"

64.4.8.137 - - [24/Jan/2007:06:42:38 -0500] "GET /courses.php HTTP/1.0" 200 7509 "-"
"msnbot/1.0 (+http://search.msn.com/msnbot.htm)"

64.4.8.137 - - [24/Jan/2007:06:50:07 -0500] "GET /whats_new.php HTTP/1.0" 200 12404
"-" "msnbot/1.0 (+http://search.msn.com/msnbot.htm)"

64.4.8.137 - - [24/Jan/2007:10:09:30 -0500] "GET /contact.php HTTP/1.0" 200 3526 "-"
"msnbot/1.0 (+http://search.msn.com/msnbot.htm)"

64.4.8.137 - - [24/Jan/2007:10:09:34 -0500] "GET /consulting.php HTTP/1.0" 200 4936
"-" "msnbot/1.0 (+http://search.msn.com/msnbot.htm)"

64.4.8.137 - - [24/Jan/2007:10:09:34 -0500] "GET /sclist.php HTTP/1.0" 200 3139 "-"
"msnbot/1.0 (+http://search.msn.com/msnbot.htm)"
```

Wow, those were ugly

So, what is missing from our logging?

- Meaningful data about the software
- Pretty much all of WWWWH

So let's see what can be done to improve things

- Let's start by looking at some case studies

Case study 1: ISP Provisioning

Web-based ISP provisioning application
- Used to open/close/modify customer accounts

Logs used to look like this

```
1.2.4.4 - - [17/Jun/2009:12:00:30 +0200] oms "GET /adsl/?
postcode=1234ZZ&housenumber=21&floor=&dsl_type=&abo=ADSLEntryPackage&PartnerID=w
ibble HTTP/1.1" 302 5 "http://www/g/adsl/check.php" "Mozilla/5.0 (Windows; U;
Windows NT 5.1; nl; rv:1.8.1.7) Gecko/20070914 Firefox/2.0.0.7 (.NET CLR
3.5.30729)" TLSv1 DHE-RSA-AES256-SHA
```

What can we learn here?
- See the problems?
- Finding fraudulent requests is not easy

And then the CSO stepped in to improve things

If a potentially fraudulent order was created, I would have to grep through Apache logs looking for a GET request for the *postal code* of where the fraudulent order was associsted.

This only works if indeed a physical address was necessary, like for a DSL line. If this was for something like UMTS, this would not exist at all. Literally, all I had was the process which checked to see if DSL was available at a particular physical address.

What is particularly frustrating above is that the IP address of the HTTP request could either be an internal system, or, the remote IP address of the customer's connection, depending on how the order was processed. Basically, this is server logging, not really application level logging.

Provisioning, cont'd

Version 1.1 looked like this

```
Jan 12 17:46:57 oms[INFO] 22.20.16.130 Order 911956 created with userid vdhmople custid 1231560
Jan 12 19:23:43 oms [INFO] 14.10.2.120 Order 911970 created with userid kpmople custid 125518
inpersonated by adsfalsvaso
Jan 12 19:40:16 oms [INFO] 9.21.162.178 Order 911978 created with userid hthmople custid 1232450
Jan 12 19:49:30 oms [INFO] 8.11.3.60 Order 911979 created with userid spemople custid 417793
Jan 12 19:52:23 oms [INFO] 8.95.14.86 Order 911982 created with userid gimople custid 291704
Jan 12 20:28:14 oms [INFO] 8.131.0.104 Order 911987 created with userid fammavoople custid 1138152
Jan 12 20:36:38 oms [INFO] 6.221.17.65 Order 911989 created with userid hartjmople custid 702782
Jan 12 20:57:24 oms [INFO] 8.80.113.101 Order 911996 created with userid jadvmple custid 1232451
```

A little better, but we're not there yet

I have the Remote IP, a unique order number, the customer ID an order was created for by name, and by number. That gives me the most flexibility with integration with other tools.

But it was still not quite enough. What I don't know from the above is *what* was ordered. I would have to go to the order management system and put in the order ID into a gui interface, query the record, and get a dump of information. Time consuming and frustrating having to use multiple systems to get the data I want.

Provisioning, cont'd

Third attempt, showing improvements

```
Feb 11 19:14:06 oms : [audit][INFO] 1.195 xso_new order 942963 created with userid murfle klantid 1240655

Feb 11 19:14:06 oms : [audit][INFO] 1.195 xso_new order 942963 created with userid murfle klantid 1240655

Feb 11 19:14:06 oms : [audit][INFO] 1.195 xso_modem order 942964 created with userid murfle klantid
1240655

Feb 11 19:14:06 oms : [audit][INFO] 1.195 bns_ship order 942965 created with userid murfle klantid
1240655

Feb 11 19:14:06 oms : [audit][INFO] 1.195 xso_modem order 942964 created with userid murfle klantid
1240655

Feb 11 19:14:06 oms : [audit][INFO] 1.195 xso_engineer order 942966 created with userid murfle klantid
1240655

Feb 11 19:14:06 oms : [audit][INFO] 1.195 xso_engineer order 942966 created with userid murfle klantid
1240655

Feb 11 19:14:06 oms : [audit][INFO] 1.195 xso_q4mobile order 942967 created with userid murfle klantid
1240655

Feb 11 19:14:06 oms : [audit][INFO] 1.195 xso_q4mobile order 942967 created with userid murfle klantid
1240655
```


Closer. Only one problem remains. The "impersonated" tag in previous slide. This is so a tech can do something on behalf of a customer. We still need to log who did that.

Provisioning, cont'd

Last (and current) version

```
Feb  5 06:57:58 oms : [audit][INFO] 1.2.3.4 voip_new order 937282 created with userid blah custid
286010

Feb  5 10:00:32 oms : [audit][INFO] 5.6.7.8 voip_new order 937402 created with userid basdflah custid
1222877 inpersonated by savdskalf

Feb  5 10:16:46 oms : [audit][INFO] 83.13.55.220 voip_new order 937418 created with userid bzavalah
custid 1216223

Feb  5 11:12:57 oms : [audit][INFO] 8.9.253.36 voip_new order 937528 created with userid xblah custid
453088

Feb  5 11:31:58 oms : [audit][INFO] 83.11.19.154 voip_new order 937543 created with userid brblah
custid 1230761

Feb  5 11:42:59 oms : [audit][INFO] 3.61.18.91 voip_new order 937555 created with userid hecblah custid
1231818

Feb  5 11:52:13 oms : [audit][INFO] 15.22.61.13 voip_new order 937567 created with userid eblah custid
441733

Feb  5 12:42:59 oms : [audit][INFO] 8.13.241.249 voip_new order 937636 created with userid jmblah
custid 1228739
```

Now we have all the WWWWH data we need

I can search, easily, for type of order, IP that placed it, order numbers, user ID's, customer identity numbers, and so on.

Provisioning, cont'd

See how the end result has improved things?

Security team can now find business-relevant data in the logs

- The logs are now written for the right audience

The log formats (syslog using log4j) are unchanged

Code considerations

What do we need to take care of in our code?

- It's NOT sufficient to change those debug statements to log4j
- Some issues require careful planning

Infrastructure first

Be sure your logging architecture is solid

- Centralized log server
  - Monitored by security team
- Secure protocols
- Non-repudiation of logged data
- Mutual authentication
  - How do you know you're talking to the logger
- What's the performance impact?
  - Separate admin data from production

Case study: Web App (servlet)

We'll step through some code issues using a Java EE servlet

- Code excerpts are meant for illustrative purposes only
- Not meant to be compilable per se

Scenario

You're the dev team leader for some enterprise code

- Boss has asked to add some functionality

New function

- Users can view their account settings

Assumptions

- Only authenticated users have access to the servlet
  - Enforced by Java EE container in presentation layer
- No input validation takes place on the client

Servlet excerpt: first attempt

```
String Lastname = request.getParameter("LASTNAME");

String Query = "SELECT Accounts FROM CUSTOMERS WHERE LASTNAME = '" + Lastname + "'";

// Query will be:   SELECT Accounts FROM CUSTOMERS WHERE LASTNAME = '_lastname_'

Statement selectStatement = connection.createStatement ();
ResultSet resultSet = selectStatement.executeQuery(Query);
```


This is functionally correct.

DISCUSSION: Is this secure? Of course not! It's vulnerable to _everything_. It is, however, functionally complete. It correctly implements the functionality requested by your boss.

Unfortunately, your boss has engaged a penetration test team, who quickly point out that your code is vulnerable to SQL injection...

Servlet excerpt: added SQLi

```
String Lastname = request.getParameter("LASTNAME"); // From HTTP request

String Query = "SELECT Accounts FROM CUSTOMERS WHERE LASTNAME = ?";

PreparedStatement pstmt = connection.preparedStatement(Query);
pstmt.setString(1, Lastname);
try
       {
                ResultSet results = pstmt.execute();
       }
```


DISCUSSION: Now is it secure? Not by a long stretch, although it is secure against SQL injection specifically. Plus, if the database contains malicious data, there is a real chance it will be output into the user's browser and Bad Things will happen. (e.g., "<script>foobar</script>")

To prevent tainted data output from doing damage, we would "escape" the data on the output side of things. This can be done via output encoding in the presentation layer .JSP file or elsewhere. I'll omit that step here, however. But output encoding into HTML context would generally change, for example, the "<" into a "&lt", the ">" into a "&gt", and so on. The characters will then _render_ correctly, but will not cause harm.

But what about tainted data input? Although our code is a simple query, what if it was inserting data into a table? We'd still have the means of injecting scripts and other harmful data. Parameterized queries do not protect against malicious data input. We must add that...

Servlet excerpt: added XSS

```
Protected final static String LEGITNAMECHARS = "^[a-zA-Z\s.\-]+$";
boolean validated = false;
String Lastname = request.getParameter("LASTNAME"); // From HTTP request

if (Lastname != null)
   {
          pattern = Pattern.compile(LEGITNAMECHARS);
          validated = pattern.matcher(Lastname).matches();
    }

if (validated) {
   String Query = "SELECT Accounts FROM CUSTOMERS WHERE LASTNAME = ?";

   PreparedStatement pstmt = connection.preparedStatement(Query);
   pstmt.setString(1, FilteredLastname);
   try
          {
                       ResultSet results = pstmt.execute();
          }

   Statement selectStatement = connection.createStatement ();
   ResultSet resultSet = selectStatement.executeQuery(sel);
}
```


Now is it secure? (Apart from pissing off the Irish and other nationalities, at least...)

It is safe, but is it ready for prime time? No, not really.

Definition: Security tiers

Levels of app security

- Tier 1
  - Block the bad stuff from happening
- Tier 2
  - Block the bad, and log
- Tier 3
  - Block, log, and take evasive actions

Our little servlet seems reasonably secure up to tier 1.

Let’s consider what it would take to go to tiers 2 and 3.

Servlet excerpt: towards tier 2

```
// We assume Log4J is already initialized and available to our code.
// (See  http://www.devdaily.com/blog/post/java/simple-log4j-example

Protected final static String LEGITNAMECHARS = "^[a-zA-Z\s.\-]+$";
String Lastname = request.getParameter("LASTNAME"); // From HTTP request
if (Lastname != null)
   {
       pattern = Pattern.compile(LEGITNAMECHARS);
       validated = pattern.matcher(Lastname).matches();
  }
if (validated)
   {
       /* Do business function */
   }
else
  {
       log.info("Attack detected!");
 }
```


How's that? Terrible. For starters, just because the regex failed doesn't mean it's an attack. What if the user simply made a mistake in the input field?

On the other hand, if we're already doing input validation in the client (e.g., a regex in Javascript in the browser), then the situation is different. Assuming the two regexes are the same, then a failure back on the server tells us that we *are* under attack -- someone has deliberately tampered with the browser Javascript. Or, perhaps their browser simply isn't accepting Javascript on our site... What to do? It depends.

Also, the log info tells us nothing at all. Consider the "who, what, when, where, and how" sorts of questions.

Servlet excerpt: closer to tier 2

```
if (validated)
   {
       /* Do business function */
   }
else
  {
       String RawLastname = Lastname;
       Lastname = RawLastname.toLowerCase();
       if Lastname.indexOf("<script>") != 0 {
          log.info("XSS attack detected!")
       }
       /* and so on */
  }
```


Better? A little, but wouldn't that list become pretty enormous? Yes, it would.

Also, the logs themselves don't contain much WWWWH detail, do they? None.

Servlet excerpt: tier 2ish

```
/*     OK, so let's consider what we log in more detail:

       Who -- We need to log the caller here.  That should come from
       somewhere outside the direct control of the user.  We'll grab
       that from the Session object.  Also src/dst IP and other
       packet-layer detail.

       What -- What did the attacker do?  Known attack vs. unknown.
       Raw data or quarantine of malicious data?

       When -- basic time/date stamp.

       Where -- See src/dst data above. */
```


What do we do with the attack data itself?

We CERTAINLY don’t want (say) XSS data in a syslog string that gets viewed in a browser...

Evidence vs. logging

Servlet excerpt: tier 3 stuff

```
/*     Possibilities, in ascending order, include (in addition to Tier two steps above):

                        Turn up logging of offending user

                        Kill session and force reauthentication

                        Quarantine the attack data

                        Null out any PII in the account

                        Kill/disable the account

                        Store attack in "evidence bag" with
                        tamper-evident seal

                        Put attacker in a "walled garden" where he can
                        do no harm (but thinks he can). */
```

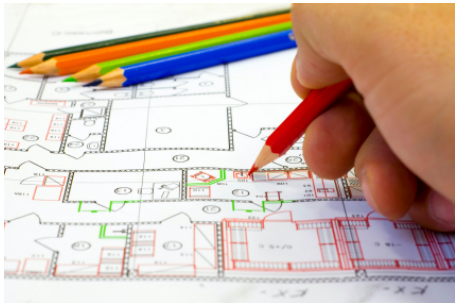

Here we have a policy conundrum. We can code various policy procedures, but need policy input. We'll provide some ideas here.

Design issues

Do we include this sort of thing everywhere, in every servlet or input?

- Centralization can make sense
- What about context?

Building some primitive functions can make sense



Candidates

Primitives to consider building

- Quarantine data (with evidentiary support)
- Common input validation attack recognition
- Evasive actions

Legacy apps

How do you improve the auditability of your legacy apps?

- Application firewalls can help to a degree
  - Most are exclusively for web apps
- Must have intimate knowledge of how the app works in order to be useful
- Event logging is a trivial and natural add-on this way

Getting started

Don’t wait for “them” to come to you

- Seek out the CSIRT at earliest stage of the dev process
  - Coordinate features, logging, etc.
  - Inventory of what gets logged is vital
  - Interface with IDS data/team to ensure compatibility with app logging data
- Seek out General Counsel or privacy officer
  - Ensure logging is in compliance
  - May need to be different by region

Kenneth R. van Wyk
KRvW Associates, LLC

ken@krvw.com

Scott McIntyre
CSO, XS4All

scott@xs4all.nl