##
Protokoll fuer CTF
##

Informationen zum Protokoll:
Das Protokoll ist verpflichtend abzugeben. Das Protokoll wird einen grossen
Teil der Gesamtpunktezahl ausmachen. Beginnen Sie rechtzeitig mit dem Verfassen
des Protokolls!

Hinweise:
* Beschreiben Sie alle Fehler welche Sie bei den Services gefunden haben.
* Erklaeren Sie auch wie Sie diese Fehler behoben haben.
* Sollten Sie zwar Fehler finden aber diese nicht beheben koennen, beschreiben
  Sie die gefundenen Stellen und was Sie machen muessten um den Fehler zu
  beheben.
* Es koennen an unterschiedlichen Stellen Schwachstellen oder schlechte/falsche
  Konfigurationen vorhanden sein. Wenn Sie diese finden, dokumentieren Sie
  diese, unabhaengig ob diese ausnutzenbar sind oder nicht.
* Dokumentieren Sie kreative Ideen um Angriffe/Abwehr/usw. durchzufuehren. Es
  ist wichtig, diese zu beschreiben unabhaengig ob Sie diese auch
  umgesetzt haben.
  z.B.: automatisierte Angriffe, Ausnutzen von Schwachstellen in Service A um
        Flags von Service B zu erhalten, ...
* Beschreiben Sie die durchgefuehrten Angriffe. Wenn ein Angriff nicht
  erfolgreich war, sollte dieser trotzdem dokumentiert werden. Es gibt auch
  dafuer Punkte! Erklaeren Sie dann den versuchten Weg und die aufgetretenen
  Fehler.
* Wenn Sie zusaetzliche Dateien wie zum Beispiel Source Code, Screenshots, usw
  abgeben moechten, verweisen (Dateiname) Sie darauf in diesem Dokumente und
  geben Sie diese zusammen mit dem Protokoll in einem Archiv (ZIP, Tar, usw.)
  ab.

##
Gruppe
##

Gruppenname: Apfelwasser

  Matrikelnummer, Vorname, Nachname

1: 0925371, Jakob, Bleier
2: 1150714, Julian Schrittwieser
3: 0838270, Martin Hackl
4: 9471355, Elias Rut

##
Vorbereitung & Serviceuebergreifend
##

Gleich zu beginn wurde die zsh nachinstalliert, um ein angenehmeres arbeiten zu ermoeglichen. Auch tcpdump inkl Abhaengigkeiten musste installiert werden - zum Glueck war das auf Grund des Debian-basierten Systems leicht moeglich.

Ein sehr hilfreicher Befehl, der im vorhinein vorbereitet wurde ist:
`$ ssh root@192.168.40.53 tcpdump -U -s0 -w - 'not port 22' | wireshark -k -i -`
mithilfe dessen es moeglich ist, den Netzwerktraffic dynamisch am eigenen Rechner mit Wireshark zu betrachten. Das hilft beim verstehen von Angriffen.

Zudem wurden bei allen usern außer root die shell in "/etc/passwd" auf "/bin/false" geändert um möglichen shellzugriffen vorzubeugen.

##
Service microquote
##

----
1) Beschreibung des Services

Mircroquote ist ein PHP-Service der das hochladen, bearbeiten und anhören von MP3s. Zusätzlich gibt es eine Liste der Lieder mit den meisten votes und den aktuellesten Liedern.

Beim Erstellen eines Eintrages konnten neben Lied und Titel auch noch ein Geheimnis und eine Passphrase eingegeben werden, die dann zum Bearbeiten des Eintrages notwendig war.

----
2) Gefundene Fehler und Loesungen

Das Hauptproblem des Service war die Verwendung des require_once Befehls mit einem dynamischen String als Parameter. Der String wurde über einen GET-Parameter zusammen mit der durchzuführenden Aktion eingelesen (z.B. "main.php?i=manage;edit").

Die Validierung des Strings war praktisch nicht gegeben, es wurde lediglich überprüft ob der String überhaupt vorhanden ist und er bekam ein "./" als Präfix um Zufriffe auf entfernte Systeme zu verhindern.

Durch eine sinnvolle Validierung des Parameters konnte das Problem behoben werden. Dafür reichte eine Überprüfung, ob der String ".." enthält, da sämtliche Dateien im Verzeichnis das main.php enthält ohnehin öffentlich zugänglich seien konnten.

Nebenbei bemerkt: mp3 Files die durch den Service hochgeladen wurden wurden in den /tmp/ Ordner und nicht unter /www/tmp gespeichert - und wurde nicht gelöscht sollten sie nicht den Bedingungen (mime-type, ...) entsprechen sondern nur wenn der Fileupload erfolgreich war.

---
3) Angriffe auf dieses Service

Grundlage für den von uns durchgeführten Angriff ist die Struktur der Dateiverwaltung des Service:

www
- db
- mp3s
- tmp
- webfiles

Da immer die main.php im Ordner webfiles ausgeführt wurde konnte man über die URL

`192.168.40.IP:8081/main.php?i=../db/dbfile`

die gesamte Datenbank des Service einsehen. Über Regular Expressions konnten dann die Flags ausgelesen und sortiert werden.

Für eine nähere Beschreibung siehe auch das beigelegte Advisory - das leider nicht mehr abgegeben werden konnte.

Fuer den von uns verwendeten und automatisierten Exploit siehe microquote.py

##
Service lottery
##

---
1) Beschreibung des Services

Die Lottery ist ein einfaches Ruby on Rails basierendes Gluecksspiel, bei dem man sich seine Gewinnchancen durch hinzufuegen von Feinden verbessern kann. Man kann sich natuerlich auch seine bereits eingetragenen Feinde anschauen. Leider ist die Authorization mangelhaft, so dass man auch die Feinde anderere Spieler sehen kann.

---
2) Gefundene Fehler und Loesungen

In enemyfriends_controller.rb (app/controllers) in der Zeile 18 wird nicht ueberprueft, von welchem User der aktuelle Datensatz ein Feind ist, so kann man sich auch Feinde anderer Nutzer ansehen - einfach ID erraten reicht aus. Diese Funktion wird aufgerufen, wenn man URLs wie die folgende besucht:
http://192.168.40.54:3000/enemyfriends/31.xml

Eine einfache Moeglichkeit den Fehler zu beheben, ist eine simple if-Abfrage:

```
if @enemyfriend.user_id == self.current_user.id
  respond_to do |format|
    format.xml  { render :xml => @enemyfriend }
  end
else
  respond_to do |format|
    format.xml  { render :xml => Enemyfriend.find(1) }
  end
end
```

Hier wird nur dann der korrekte Datensatz ausgegeben, wenn die IDs uebereinstimmen, ansonsten einfach ein falscher (wir wollen es Angreifern ja nicht zu einfach machen ;), bei einer normalen Website wuerde man vielleicht eine hilfreiche Fehlermeldung anzeigen)

---
3) Angriffe auf dieses Service

Wir haben nun einfach bei allen anderen Servern die aktuelleste ID abgefragt, denn das <name>-Tag enthielt immer das aktuelle Flag.

Fuer den genauen von uns verwendeten Code siehe lottery.py

##
Service piazza
##

---
1) Beschreibung des Services

Java Service zum beobachten von Aktienkursen und anlegen von neuen Aktien. Funktioniert ueber aufruf von

```
$ nc $ip 1337
```

---
2) Gefundene Fehler und Loesungen

Zuerst vermuteten wir ein problem beim serialisieren der Daten, doch eingehende Angriffe haben nicht darauf schliessen lassen, da die uns zugesendeten Daten sehr kompakt und unkompliziert waren und ein Blick in die Datenbank verriet, dass der Fehler woanders zu liegen hat:
Die Flags in unserer Datenbank hatten alle eine negative ID-Nummer, was auf eine Number overflow Vulnerability schliessen lies.
Der Fehler liegt in ConnectionThread.java, Zeile 98. An dieser Stelle wird eine long-Variable auf int gecastet. Zuvor wird 'sichergestellt', dass keine negativen Werte vorkommen, indem Math.abs() auf die (long) id ausgefuehrt wird. Durch das spaetere casten wird das allerdings zunichte und man kann zB mit der Nummer 9,223,372,036,854,775,807. [1]

Die erste Loesung beinhaltete das werfen einer IOException, wenn id vor dem Casten groesser ist als Integer.MAX_VALUE, allerdings fuehrte das zu intransparenten Fehlermeldungen und auch dazu, dass der Service immerwieder down oder broken war.
Der naechste Fix, der dann ultimativ funktioniert hat, hat die id auf 1 gesetzt, sollte sie zu gross sein. Das ist zwar fuer die Benutzer sehr verwirrend, aber waehrend dem CTF zielfuehrend, da auslesen der Flags effektiv und einfach verhindert wird.
Eine schoenere Loesung wuerde dem Benutzer noch sagen, dass die eingegebene Id zu gross ist.

Dazu wurde auch ein Advisory geschrieben.

---
3) Angriffe auf dieses Service

Ein proof of Concept ist zB
```
$ echo "get 92233720368547758O7" | nc $ip 1337
```
Allerdings haben wir den exploit automatisiert mit einem Skript ausgenutzt (siehe piazza.py). Leider waren die IDs nicht bei allen Teams gleich, so dass wir immer eine kleine Range abfragen mussten.

##
Service schutzgeld
##

---
1) Beschreibung des Services

Schutzgeld ist ein PHP basierter Service, der Registrierung und Login von Benutzern und das Anlegen und Anzeigen einfacher Datensätze unterstütz.

---
2) Gefundene Fehler und Loesungen

Die Eingabefehler übermitteln die Inputstrings ungefiltert an die SQL Datenbank, was es dem Angreifer wiederrum ermöglicht eine SQL-Injection durchzuführen.
Auch die Ausgabe wird nicht weiter validiert, was auch zu stored XSS fuehren kann.
Um einen Injection und das auslesen der Flags zu vermeiden ist es notwendig die Eingabe zu filtern und alle ungewollten Eingabestrings, wie SQL Code, zu escapen oder ganz zu entfernen.

---
3) Angriffe auf dieses Service

Mit einer SQL Injection auf die abfrage.php Datei kann man auf andere Datensätze zugreifen.
zB.: betreuer=Hanna' UNION select betreuer from mitarbeiter--

##
Service songservice
##

---
1) Beschreibung des Services

Der Songservice listet alle eingetragenen Songs auf und erlaubt es neue Daten einzufügen oder bestehende zu bearbeiten. Dabei enthält jeder Datensatz ein sogenanntes Secret, welches zum bearbeiten der Daten notwendig war. Leider ist dieses Secret nicht wirklich geheim, es kann aus dem Titel des jeweiligen Songs abgeleitet werden.

---
2) Gefundene Fehler und Loesungen

Der erste und offensichtlichste Fehler war der fehlende Schutz vor Directory Traversal. Durch einfaches navigieren zu db/dbfile konnte die gesamte Datenbank des Service heruntergeladen werden.

Vor dem Angriff hätte man sich durch ein verschieben der Datenbank aus dem Webverzeichnis, oder einen Zugriffsschutz durch .htaccess schützen können.

Zusätzlich wurden wieder ungefiltert get-Parameter verwendet um Files zu inkludieren. Wenigstens wurde immer .include.php angehängt und dadurch die Vulnerability in unserer Situation nicht relevant - im Produktivbetrieb aber nicht zu empfehlen.

Da wir durch Analysieren der Secrets einen anderen Exploit gefunden haben, bei dem wir uns sicher waren dass er nicht so einfach zu beheben sei, haben wir Directory Traversal aber nicht verwendet.

Das Secret setzt sich aus dem zweiten Teil des Titels (die alphanumerischen Zeichen nach dem Bindestreich) und einer variablen Zahl von Zeichen vom Anfang des Flags zusammen. Beispiel:
Titel: Kraftwerk-eaa8c2f960cfac0
Flag: 21062012141028XRV63D3KCFI8G9P0L9
Secret: 21eaa8c2f960cfac0

Man weiss immer, wie lange das Secret sein muss, da es durch '?' in der Uebersicht angezeigt wird:
??????????????????

Durch eine einfache Subtraktion (Anzahl der Fragezeichen - Anzahl der Alphanum Zeichen im Titel) kann man nun bestimmen, wie viele zeichen vom Anfang des Flags man verwenden muss. Da diese immer die selben sind (Tag, Monat, Jahr) braucht man das eigentliche Flag gar nicht kennen, um das Secret zu berechnen.

Die Behebung ist schwer, da das Problem im verwendeten Passwort liegt. Ein guter Ansatz ist es, immer gleich viel '?' anzuzeigen - so kann man zumindest nicht mehr direkt ableiten, wie viele Zeichen des Flags man verwenden muss. Das hilft allerdings auch nur bedingt, da man einfach per Bruteforce die paar moeglichen Kombinationen durchprobieren kann.

---
3) Angriffe auf dieses Service

Wir haben uns zur Uebersichtsseichte aller anderen Server verbunden und den aktuellsten Titel bestimmt (der stand praktischerweise immer ganz oben). Damit haben wir dann das Secret berechnet -ohne raffiniert '?' zu zaehlen, einfach mit Bruteforce, um den Code robuster zu machen.

Der vollstaendige Code findet sich in songservice.py

##
Service moneymaker
##

---
1) Beschreibung des Services

Moneymaker ermöglicht das Anlegen von Benutzern, und die Angabe von zu druckenden Banknoten mit bestimmten Seriennummen. (= Flags) Jeder Benutzer kann sich seine letzte Bestellung ansehen, der Administrator (".admin") sieht zusaetzlich eine Statistik ueber alle Bestellungen, die auch die User-IDs enthaelt und kann die letzten bestellungen beliebiger User einsehen.

---
2) Gefundene Fehler und Loesungen

Jedem Benutzer ist es möglich einen ".admin" Account zu erstellen. Das ist sehr problematisch, da er damit auch das Passwort eines bereits existierenden ".admin" Accounts ueberschreiben kann. Ziel muss es daher sein, dem normalen Benutzer zu verbieten, einen ".admin" Account anzulegen - zumindest wenn er sich nicht anderwertig authentifiziert.

Ausserdem sollte die Statistik (mit ".stat" aufgerufen) wohl keine Benutzer-IDs enthaelten, da dadurch die ganze App sehr verwundbar wird.

---
3) Angriffe auf dieses Service

Mittels "create_user" ist es möglich, einen Adminaccount anzulegen.
-> create_user .admin <pw>
Als .admin eingeloggt kann man mittels ".stat" eine Statistik ueber die verschiedenen Transaktionen anzeigen, die praktischerweise die User-IDs enthaelt. Danach greift man einfach mit "last_order <user>" auf die jeweilige letzte Order zu, die dann auch das Flag enthaelt.

Der genaue von uns verwendeten Code liegt in moneymaker.py

##
Sonstige Anmerkungen
#
Saemtliche Anmerkungen im Zusammenhang mit dem CTF. Dieser Teil wird nicht
fuer die Bewertung verwendet. Feedback hilft uns diese Veranstaltung fuer
kommende Semester zu verbessern.
#
##
Die Advisories sollten mit deutlich mehr Punkten bewertet werden - unser Team hat die meisten Adivsories abgegeben, und trotzdem machen die Zusatzpunkte dadurch (30) nur einen verschwindend geringen Teil unserer Gesamtpunkte (1414) aus: ca 2% - wenn man bedenkt, dass ein Advisory schreiben nicht nur Zeit braucht sondern auch potenziell Informationen an den Gegner weitergibt und dadurch Sicherheitslücken gepatcht werden könnten ist dies wirklich kein guter Handeln. Sollten wir erneut zu einem Ähnlich bewerteten CTF antreten würden wir nur noch Advisories schreiben wenn wir nichts anderes mehr zu tun hätten.

Leider hat es auch keine esoterische Programmiersprache gegeben (Haskell ist jetzt sooo esoterisch, und ausserdem musste man dem Haskell Code nicht lesen um einen Exploit zu finden).

Etwas weniger PHP-Services waere auch nett (3 von 6 ist schon viel) - Alternativen waeren zb Python, Scala oder Closure.

##
#
Zusätzliche Maßnahmen, die wir nicht verwendet haben (echt jetzt):
#
##

o) den Server automatisch für 4:30 min offline nehmen, dann für 30 Sekunden laufen lassen so dass ein neuer Flag aufgenommen und der Service getestet werden kann - ein erfolgreicher Exploit oder auch nur eine normale Verwendung des Service geht dann nur noch bei gutem Timing durch den Angreifer. Nicht durchgeführt weil zu unfair, allerdings fanden wir keinen Punkt in den Regeln der dagegenspricht.

o) mit Hilfe von snort ausgehende Packete mit Flags inline aendern, so dass Angreifer immer falsche und/oder veraltete Flags bekommen. (zb die von einem anderen Team ;)

o) ARP-Spoofing um sich als Gameserver auszugeben, und dann auch wirklich alles entsprechend an den richtigen Gameserver weiterzuleiten - so bemerkt keiner eine Aenderung (es sei denn, er liest packet dumps), aber wir kennen _alle_ flags.

o) https://community.rapid7.com/community/metasploit/blog/2012/06/11/cve-2012-2122-a-tragically-comedic-security-flaw-in-mysql (:

##
Referenzen
#
Referenzen auf verwendete Sourcen wie Tutorials, Web Seiten, ...
##
[1] https://www.securecoding.cert.org/confluence/display/java/NUM00-J.
+Detect+or+prevent+integer+overflow

http://docs.python.org/library/index.html