В данной главе мы обсудим с Вами анализ исходного кода веб-приложений на наличие в нём ошибок связанных с нарушением безопасности. Такой поиск уязвимостей сильно отличается от анализа сайта через браузер, но и найти таким способом можно намного больше уязвимостей. Тем более часто встречаются сильно запутанные уязвимости и начинает казаться что вроде бы уязвимость есть, а вроде бы её и нет. Тогда остаётся только одно – лезть в код и искать ответ там.

Мы рассмотрим несколько уязвимостей и напишем для них эксплойты. Уязвимости я старался подобрать не обычные, а редкие или просто интересные. Но для начала обсудим то с чем Вы можете столкнуться при анализа кода.

Вопервых если Вы решили анализировать любой код Вам нужно запастись терпением. Дело это очень изнурительное и сильно выматывающее. По своему опыту могу сказать что иногда приходится по 2-3 дня сидеть и копаться в коде без каких-либо результатов. Иногда может попадаться код просто плохо читаемый или очень "коряво" написанный. В последнем случае лучше сразу отказаться от анализа такого приложения если конечно у Вас нет какой-то особой цели. Но со временем ко всем этим обстоятельствам (кроме последнего) и неудобствам можно привыкнуть. К тому же анализ кода очень хороший источник идей, опыта и каких-либо интересных приёмов программирования.

Если рассуждать глобально то есть два основных метода анализа кода – простой поиск уязвимых мест по какому-либо шаблону и логический поиск уязвимостей. Первый вариант очень плохой, но большинство новичков именно с него и начинают. Суть данного метода сводится к обычному поиску SQL-запросов, функций подключений файлов и функций выполнения PHP-кода. При обнаружении такого кода человек просто пытается найти способ вставки своих данных, например, в SQL-код обнаруженного запроса. Грубо говоря весь поиск уязвимостей этим методом сводится к поиску строк похожих на эти:

```
include($_GET['...']);
$sql = "SELECT .... FROM ... WHERE id=".$_GET['id'];
eval(.....$_POST['theme'].....);
```

То есть при таком методе анализа мозг отключается если не полностью, то процентов на 90 точно. Вы просто ищите подозрительные места даже не вникая в саму работу кода. Если Вы таким методом найдёте какую-то уязвимость то в большинстве случаев Вы очень много времени затратите на нахождение способа её эксплуатации. Но этот вариант имеет и свои плюсы. Например, его можно использовать для быстрого анализа на самые очевидные уязвимости.

Логический метод анализа кода намного сложнее (сложность зависит от Ваших знаний), но в разы эффективнее. Его суть заключается в том что бы понять внутреннее устройство анализируемого кода и разобрать его полностью. Вот как раз при таком варианте и можно узнать для себя много нового. Но если для простого поиска уязвимых мест нужна в основном внимательность, то для логического поиска уязвимостей весь упор делается на знания PHP. Возможно что в первое время при таком анализе придётся обложится книгами и справочниками по PHP и детально изучать каждый подозрительный кусок кода или неизвестную Вам функцию. Как пример уязвимости найденной методом логического анализа можно рассмотреть ошибку проверки входящих данных в движке WCSP которую обнаружила наша команда. Вот уязвимый код:

```
if ( is_array($_POST) && strstr($_SERVER['PHP_SELF'],'admin.php')) {
  // здесь код, который фильтрует $_POST для администратора
}
elseif ( is_array($_POST)) {
   // здесь код, который фильтрует $_POST для всех
}
elseif ( is_array($_COOKIE)) {
  // здесь код который фильтрует $_COOKIE
}
```

В чём же суть уязвимости? Давайте разберём код по порядку, со стороны обычного пользовательского обращения к сайту. Первое условие сработает только тогда когда скрипт, к которому обращаются имеет имя "admin.php". Так как мы действуем со стороны обычного пользователя то это условие можно отсечь, остаются ещё два:

```
elseif ( is_array($_POST)) {
   // здесь код, который фильтрует $_POST для всех
}
elseif ( is_array($_COOKIE)) {
  // а здесь типа фильтрует $_COOKIE =)
}
```

И вот тут как раз и имеется уязвимость. Данный код закончит свою работу именно на втором условии потому что при любом стечении обстоятельств массив $_POST всегда будет массивом, даже если нет никакой информации переданной POST-методом. Следовательно до фильтрации массива $_COOKIE дело вообще никогда не дойдёт. Поэтому становятся возможны SQL-инъекции через данные переданные в COOKIES и прочие уязвимости для эксплуатации которых нужно полное отсутствие их фильтрации. Что бы убедится в этом попробуйте запустить у себя вот такой код:

```
if ( is_array($_POST)) {
   print "Дошли до \$_POST";
```

```
}
elseif ( is_array($_COOKIE)) {
        print "Дошли до \$_COOKIE";
}
```

После его выполнения Вы увидите только одну надпись – "Дошли до $_POST". Я сомневаюсь что подобную уязвимость можно найти просто бегло просматривая код. Тем более логический анализ кода намного интереснее.

В ходе обсуждения уязвимостей Вам понадобятся исходные коды уязвимых версий CMS. Их Вы можете взять на диске в папке /CMS/.